

Tip #31 from

Laura

To move all four sides of the split point to the intersection of the vertical and horizontal bars. When the mouse pointer turns into a four-headed arrow, click and drag to move the intersection.

To remove the split, choose one of the following methods:

- Choose **Window, Remove Split**. All split bars will be removed.
- Double-click one split bar to remove it. To remove a four-way split, double-click the intersection of the two splits.
- Drag one of the split bars off the worksheet.

Tip #32 from

Laura

Use the **split boxes**, small buttons at the top of the vertical scrollbar and to the right of the horizontal scrollbar, to insert and move split bars. If you can't see the split boxes, unfreeze the window; you can't split a frozen window.

HIDING AND UNHIDING ROWS AND COLUMNS

The word *hiding* can give the impression that something sneaky is happening, that something is being kept a secret. That may be your motive for hiding a column or row in your worksheet, but it's probably not the only one. Hiding rows and/or columns allows you to keep something from being printed if the content is of no interest to the person who'll be reading the printout (that may even be you), or to simplify the view of the worksheet, removing distracting or visually cluttering content while you work.

Tip #33 from

Laura

Hiding content also can be used to make confidential content invisible, but there are better ways to do that, such as password-protecting a workbook or placing the file in a network drive to which only you have access.

To hide a column or row, you can choose from the following two methods:

- Resize the column or row until it is so narrow that it literally disappears.
- Select the row(s) or column(s) and choose **Format, Column (or Row), Hide**, or right-click the selection and choose **Hide** from the context menu.

When a column or row is hidden, a thick border appears between the headings of the visible rows or columns where the hidden number or letter would normally appear. Figure 2.35 shows the special split two-headed arrow that only appears on a boundary where one or more rows or columns are hidden.

Special Edition
Using Microsoft
Excel 2000
© 1999 by Que Corp.

BEST AVAILABLE COPY

BEST AVAILABLE COPY

Split two-headed arrow mouse pointer

Project Name	Revision	Date	Page
Department	Version	3	5 and
Budget hrs to complete	Efficiency		
1200	100%	1200	100%

To reveal hidden columns or rows, choose one of the following methods:

- Point to the thick boundary between column letters or row numbers. When the mouse pointer turns into a split two-headed arrow, double-click (refer to Figure 2.34).
- Select the columns or rows that appear on either side of the hidden content, and choose **Format, Column (or Row), Unhide**.

Observe the two-headed arrow carefully—a solid two-headed arrow is used for resizing columns and rows; a split two-headed arrow is used for unhiding a column or row.

→ For more information on using protection to control changes to your workbooks, see "Protecting Your Data," p. 520

TROUBLESHOOTING

DISTINGUISHING ONE VERSION OF A FILE FROM ANOTHER

How do I determine which file is which?
Aside from...

Aside from doing a visual check for differences in content, you can distinguish two different versions of an Excel file by checking the date and time modified for each file. You can see this information in the Open dialog box, Windows Explorer, or My Computer (in Details view), or by choosing File, Properties when the workbook is active onscreen. The Statistics tab in the Properties dialog box shows the date and time the file was last modified.

CREATING A TEMPLATE FROM AN EXISTING WORKBOOK

What if I've already added data to the file that I want to use as a template?
Before you save the file as a template, save it as a new file. Then delete any specific data.

Before you save the file as a template, save it one last time in its current format. Then delete any specific data, leaving only the labels, formulas, and any other text that you want to have on every worksheet created with the template. When you save the workbook as a template, the original file, prior to the deletions, will be left intact.

WHY SAVE TO THE TEI

I've placed my template
 New dialog box. With
 Only templates that
 Spreadsheet Solution
 workbook based on
 copy or move the u
 at Windows Applic
 where they are, use
 with its new workb

EXCEL IN PRAĆ

You can use several basic cell content templates, prenamed sheets for use of the templates, and levels of template protection.

Column b

CNG Alteration
 8110 Purdue Road
 Indianapolis, IN 46226
 (317) 875-5157

Sheet tabs are name

White Paper: A Grid Monitoring Service Architecture (DRAFT)

*Brian Tierney, Ruth Aydt, Dan Gunter, Warren Smith,
Valerie Taylor, Rich Wolski, Martin Swamy, and the
Grid Performance Working Group
Global Grid Forum*

Abstract

Large distributed systems such as Computational and Data Grids require a substantial amount of monitoring data be collected for a variety of tasks such as fault detection, performance analysis, performance tuning, performance prediction, and scheduling. Some tools are currently available and others are being developed for collecting and forwarding this data. The goal of this paper is to describe a common architecture with all the major components and their essential interactions in just enough detail that Grid Monitoring systems that follow the architecture described can easily devise common APIs and wire protocols. To aid implementation, we also discuss the performance characteristics of a Grid Monitoring system and identify areas that are critical to proper functioning of the system.

1.0 Introduction

The ability to monitor and manage distributed computing components is critical for enabling high-performance distributed computing. Monitoring data is needed to determine the source of performance problems and to tune the system and application for better performance. Fault detection and recovery mechanisms need monitoring data to determine if a server is down, and whether to restart the server or redirect service requests elsewhere [14][10]. A performance prediction service might use monitoring data as inputs for a prediction model [16], which would in turn be used by a scheduler to determine which resources to use.

There are several groups that are developing Grid monitoring systems to address this problem [11][16][9][14] and these groups have recently seen a need to interoperate. In order to facilitate this, we have developed an architecture of monitoring components. A Grid monitoring system is differentiated from a general monitoring system in that it must be scalable across wide-area networks, and include a wide range of heterogeneous resources. It must also be integrated with other Grid middleware in terms of naming and security issues. We believe the Grid Monitoring Architecture (GMA) described here addresses these concerns and is sufficiently general that it could be adapted for use in distributed environments other than the Grid. For example, it could be used with large compute farms or clusters that require constant monitoring to ensure all nodes are running correctly.

2.0 Design Considerations

With the potential for thousands of resources at geographically different sites and tens-of-thousands of simultaneous Grid users, it is important for the data management and collection facilities to scale while, at the same time, protecting the data from spoiling.

In order to allow scalability in both the administration and performance impact of such a system, the decision-making as to what is monitored, measurement frequency, and how the data is made available to the public must be widely distributed and dynamic. Thus, instead of a centralized management component, multiple independent management components synchronize their state through a directory service, which may itself be distributed. Distributing management in this fashion also helps minimize the effects of host and network failure, making the system more robust under precisely the kinds of conditions it is trying to detect.

In some models, such as the CORBA Event Service, all communication flows through a central component, which represents a potential bottleneck. In contrast, we propose that performance event data, which makes up the majority of the communication traffic, should travel directly from the producers of the data to the consumers of the data. In this way, individual producer/consumer pairs can do "impedance matching" based on negotiated requirements, and the amount of data flowing through the system can be controlled in a precise

February 27, 2001

and localized fashion based on current load considerations. The design also allows for replication and reduction of event data at intermediate components acting as consumer/producer caches or filters. Use of these intermediate components lessens the load on producers of event data that is of interest to many consumers, with subsequent reductions in the network traffic, as the intermediaries can be placed "near" the data consumers. The directory service contains only metadata about the performance events and system components and is accessed relatively infrequently, reducing the chance that it would be a bottleneck.

We also considered a purely SNMP-based solution for monitoring, but rejected it because we felt that the SNMP's simple GET/SET model is not rich enough, as there is no support for subscription. Also, it is not clear that security model maps well to the Grid Security Infrastructure. However, we definitely envision the use of SNMP-based tools as a source of monitoring data.

3.0 Architecture

The GMA architecture supports both a producer/consumer model, similar to several existing Event Service systems such as the CORBA Event Service [1], and a query/response model. For either model, producers or consumers that accept connections publish their existence in a directory service. Consumers use the directory service to locate one or more producers generating the type of event data they are interested in. Each consumer then subscribes to or queries the matching producer(s) directly. Likewise, a producer may query the directory service to locate consumer(s) that accept and process event data in a given manner – for example, a consumer that archives event data for later analysis. Once the appropriate consumer is identified, the producer would connect to it directly and stream the event data – similar in behavior to when a consumer subscribes to a producer, but initiated by the producer.

3.1 Terminology

The monitoring data that the GMA is designed to handle are timestamped *events*. An event is a named collection of data. The data may relate to anything, but common events will be memory usage, network usage, or "error" conditions such as a server process crashing. The *producer* is the component that makes the event data available. A *consumer* is any process that requests or accepts event data. A *directory service* is used to publish what event data is available and which producer to contact to get it. All of these components are described in detail below.

3.2 Components

The architecture consists of the following components, shown in Figure 1:

- consumers
- producers
- directory service

By defining three interfaces: the consumer to producer interface, the consumer to directory service interface, and the producer to directory service interface; we can build "standard" grid monitoring services that will all inter-operate.

Directory Service

To locate, name, and describe the structural characteristics of any data available to the Grid, a distributed directory service for publishing this information must be available. The primary purpose of this directory service is to allow information consumers (users, visualization tools, programs and resource schedulers) to discover and understand the characteristics of the information that is available. In addition, information producers must be

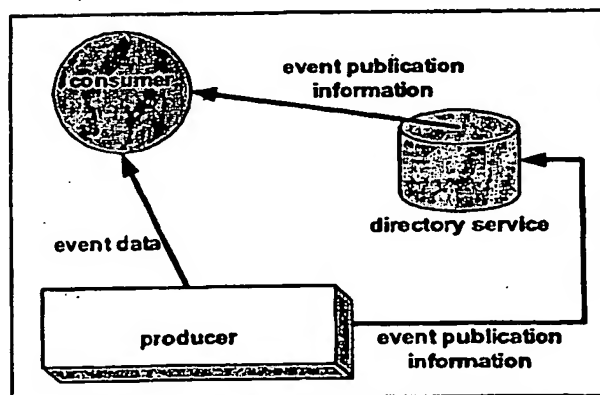


Figure 1: Grid Monitoring Architecture Components

able to update the information to reflect the system state. In the context of common operations for both consumers and producers, they will be collectively referred to as *clients*.

The directory service contains a listing of all available event data and their associated producers. This allows consumers to discover what event data are currently available (through the producer registration), what the characteristics of the data are, and which producer) to contact to receive a given type of event data.

The directory service, however, is not responsible for the storage of performance data itself—only its name and other characteristics. We assume the names and characteristics associated with dynamic performance data change slowly (unlike the performance data itself). That is, the name and structural characteristics of a data set remain relatively constant while the valid contents of the data set may change dramatically over time.

The functions supported by a directory service are:

1. **Authorize-consumer** – Establish identity of a consumer, which is in turn mapped to access permissions for the next, or possibly several subsequent, transaction(s).
2. **Authorize-producer** – <same as Authorize-consumer>, although different mechanisms may be used for the two authorization operations.
3. **Search** – Perform a search for event data. The client should indicate whether only one result, or more than one result, if available, should be returned. An optional extension would allow the client to get multiple results one element at a time using a “get next” query in subsequent searches.
 - **Preconditions:** The client is authorized to perform the search.
 - **Postconditions:** The result(s) of the search are returned, which includes a well-defined null value for searches which did not match in the directory.
4. **Add** – Add a record to the directory.
 - **Preconditions:** The client is authorized to add the record. The record conforms to the directory's schema. The record is not a duplicate.
 - **Postconditions:** The record is in the directory.
5. **Remove** – Remove a record from the directory.
 - **Preconditions:** The client is authorized to remove the record. The record matches exactly one record in the directory.
 - **Postconditions:** The record is not in the directory.
6. **Update** – Change the state of a record in the directory.
 - **Preconditions:** The client is authorized to modify the record. The record matches exactly one record in the directory.
 - **Postconditions:** The record now has the new values.
7. **Version request** – A client may request the current version of the interface. The version numbering system is TBD.

Query-optimized directory services such as LDAP [15], Globus MDS [3], the Legion Information Base, and the Novell NDS, all provide the necessary base functionality for this service, but only in their fully distributed implementations. Some public-domain implementations of these services do not support distributed implementation.

consumer

A consumer is any program that receives event data from a producer. Consumers that will accept asynchronous requests from producers will publish this information in the directory service. The functions supported by a consumer are:

1. **Authorize to producer** – The consumer contacts a producer and proves its identity. This may need to be performed once per “session”, or on every request.
2. **Authorize from producer** – The consumer accepts authorization requests from a producer and verifies its identity. As in *Authorize to producer*, this may be done once per session or on every producer-initiated request.

3. **Query** – The consumer receives one event or set of events from the producer. Optional extensions are a *filter* to indicate interest in only a subset of events or to perform transformations on event data.
 - **Preconditions:** The consumer is authorized to receive these event(s). The event data is available.
 - **Postconditions:** One or more events are returned, together, in the reply.
4. **Consumer-initiated Subscribe** – The consumer establishes a connection to the producer to receive events in a stream.
 - **Preconditions:** The consumer is authorized to connect to the producer and receive these event(s). The event data is available.
 - **Postconditions:** Same as for Query, except that in addition to returning the most recent event, on success the producer will either (a) return events in a stream over the connection used for the request or (b) inform the consumer of the location of a new connection from which it can read the stream of events.
 - **Other behaviors:** If the consumer closes the established connection, the producer should simply consider the subscription ended (generating no errors). If the underlying source of event data stops producing data, the producer may close the connection without warning, so consumers should be designed to recover gracefully in this instance.
5. **Consumer-initiated Unsubscribe** – The consumer tells a producer to close the subscription. An optional extension is a “close all” version which closes all subscriptions for this consumer.
 - **Preconditions:** The subscription exists for the producer/consumer pair. The consumer is authorized to end it.
 - **Postconditions:** The subscription is removed. No more data should be sent for this subscription after the producer has confirmed.
6. **Producer-initiated Subscribe** – The consumer accepts subscriptions from producers who wish to send events.
 - **Preconditions:** The producer is authorized to send events to this consumer.
 - **Postconditions:** A new subscription is created for this producer/consumer pair.
7. **Producer-initiated Unsubscribe** – The consumer accepts an unsubscribe request from the producer.
 - **Preconditions:** The subscription exists. The producer is authorized to end it.
 - **Postconditions:** The subscription is removed.
8. **Authorize to directory** – The consumer contacts the directory service and proves its identity. This may need to be performed once per “session” or on every lookup.
9. **Lookup** – The consumer makes a query to the directory service, of which at least 2 types should be available: (1) producer: get data for a producer associated with an event. (2) event: get the description of the event.
 - **Preconditions:** Authorization has been performed.
 - **Postconditions:** The directory service is unchanged (read-only operation).
10. **Update** – The consumer updates records in the directory service regarding events for which this consumer will accept producer-initiated subscriptions.
 - **Preconditions:** Authorization has been performed.
 - **Postconditions:** The directory service has more/less/modified records reflecting the new information.

There are many possible types of consumers. These may include:

- **real-time monitor:** This consumer is used to collect monitoring data in real time for use by real-time analysis tools. It checks the directory service to see what data is available, and then “subscribes” to all the events it is interested in. The producers then send the event data to the consumer as it is generated. Data from many sources can then be used for real-time performance analysis.
- **archiver:** This consumer may be used as to collect data for the archive service. It subscribes to the producers, collects the event data, and places it in the archive. We note that a monitoring architecture needs this component, as it is important to archive event data in order to provide the ability to do historical analysis of system performance, and determine when/where changes occurred. While it may not be

desirable to archive all monitoring data, it is desirable to archive a good sampling of both "normal" and "abnormal" system operation, so that when problems arise it is possible to compare the current system to a previously working system. In this architecture, the archive is just another consumer.

- **process monitor:** This consumer can be used to trigger an action based on an event from a server process. For example, it might run a script to restart the processes, send email to a system administrator, call a pager, etc.
- **overview monitor:** This consumer collects events from several sources, and uses the combined information to make some decision that could not be made on the basis of data from only one host. For example, one may want to trigger a page to a system administrator at 2 A.M. only if both the primary and backup servers are down.

producer

Producers are responsible for providing event data to consumers, either by request or asynchronously. Producers will publish event availability information in the directory service. The functions supported by a producer are:

1. **Authorize from consumer** – The producer establishes a consumer's identity and access permissions. Authorization may be combined with subscription or query requests, or performed separately with the results stored in a shared "key" of some sort.
2. **Authorize to consumer** – The producer contacts a consumer and proves its identity. As for *Authorize to producer*, this may need to be performed once per session or on every new request.
3. **Query** – The producer returns a single set of event(s) in response to a consumer query.
 - Preconditions: The consumer is authorized to receive data about the event.
 - Postconditions: The event data, if present, is returned.
4. **Consumer-initiated Subscribe** – Accept consumer requests to establish a stream of event data (subscription). This request should include parameters and filters, etc.
 - Preconditions: Consumer is authorized to subscribe to requested event data.
 - Postconditions: The subscription is added for a consumer, and the producer either (a) returns events in a stream over the connection used for the request or (b) informs the consumer of the location of a new connection from which it can read the stream of events.
5. **Consumer-initiated Unsubscribe** – This is the normal operation by which a consumer ends its subscription. An optional "unsubscribe all" extension would allow the consumer to cancel all its subscriptions at once. As mentioned in the consumer section, if a consumer summarily closes its connection, the producer should automatically unsubscribe it everywhere.
 - Preconditions: The subscription exists for this producer/consumer pair.
 - Postconditions: The consumer/producer pair has one less subscription.
6. **Producer-initiated Subscribe** – A producer asynchronously begins a subscription with a consumer.
 - Preconditions: The producer is authorized to send data to the consumer.
 - Postconditions: The subscription is added and the producer may now send data.
7. **Producer-initiated Unsubscribe** – The producer informs a consumer that the subscription is ending.
 - Preconditions: The subscription exists for this consumer/producer pair. The consumer supports this function, allowing producers to asynchronously unsubscribe.
 - Postconditions: The subscription is removed. Note that even in the case of failure the subscription may be removed by the producer.
8. **Version** – A consumer may request the current version of the interface. The version numbering system is TBD.

Producers can service "streaming" or "query" requests from consumers. In streaming mode the consumer makes a single request, then receives events in a stream until an explicit action is taken to end the connection. In query mode the consumer makes a single request and receives a single event in reply.

The producers are also used to provide access control to the event data, allowing different access to different classes of users. Since Grids typically have multiple organizations controlling the resources being monitored there may be different access policies (firewalls possibly), support for different frequencies of measurement, and willingness to allow access to different performance details for consumers "inside" or "outside" of the organization running the resource. Some sites may allow internal access to real-time event streams, while providing only summary data off-site. The producers would enforce these types of policy decisions. This mechanism is especially important for monitoring clusters or computer farms, where there may be a large amount of internal monitoring, but only a limited amount of monitoring data accessible to the Grid.

There may also be components that are both consumers and producers. For example a consumer might collect event data from several producers, and then use that data to generate a new derived event data type, which is then made available to other consumers, as shown in Figure2.

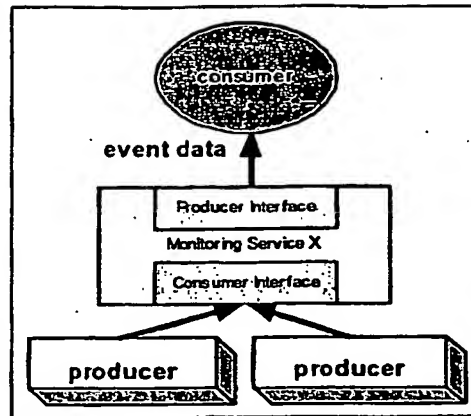


Figure 2: Joint Consumer/Producer

Sources of Event Data

There are many possible sources of event data, including monitoring *sensors*. The following is a summary of common types of sensors:

- **host sensors:** These sensors perform host monitoring tasks, such as monitoring CPU load, available memory, or TCP retransmissions. Host sensors may be layered on top of SNMP-based tools, and therefore run remotely from the host being monitored. Host sensors could also be used to monitor host configuration information, such as what versions of the operating system or other software packages are installed.
- **network sensors:** These sensors perform SNMP queries to a network device, typically a router or switch. Information on which device statistics are being monitored is published in the directory service.
- **process sensors:** Process sensors generate events when there is a change in process status (for example, when it starts, dies normally, or dies abnormally). They might also generate an event if some dynamic threshold is reached (for example, if the average number of users over a certain time period exceeds a given threshold).
- **application sensors:** Autonomous sensors can also be embedded inside of applications. These sensors might generate events if a static threshold is reached (for example, if the number of locks taken exceeds a threshold), upon user connect/disconnect or change of password, upon receipt of a UNIX signal, or upon any other user-defined event. Application sensors can also be used to collect detailed monitoring data about the application to be used for performance analysis. These types of sensors may not register themselves with the directory service, but could still feed their results to the system. A special case of application sensors would be library sensors that would be embedded in library code and compiled into the application.
- **storage or I/O sensors:** These sensors perform any monitoring of storage systems such as disks and tapes, obtaining information on block size, access time, seek time, etc.

- **middleware sensors:** These sensors would gather information about middleware services such as directory and authentication servers. They could report request volume, average service time, number of requests returned due to timeouts, etc. and would be used to discover and repair performance problems in this service layer.

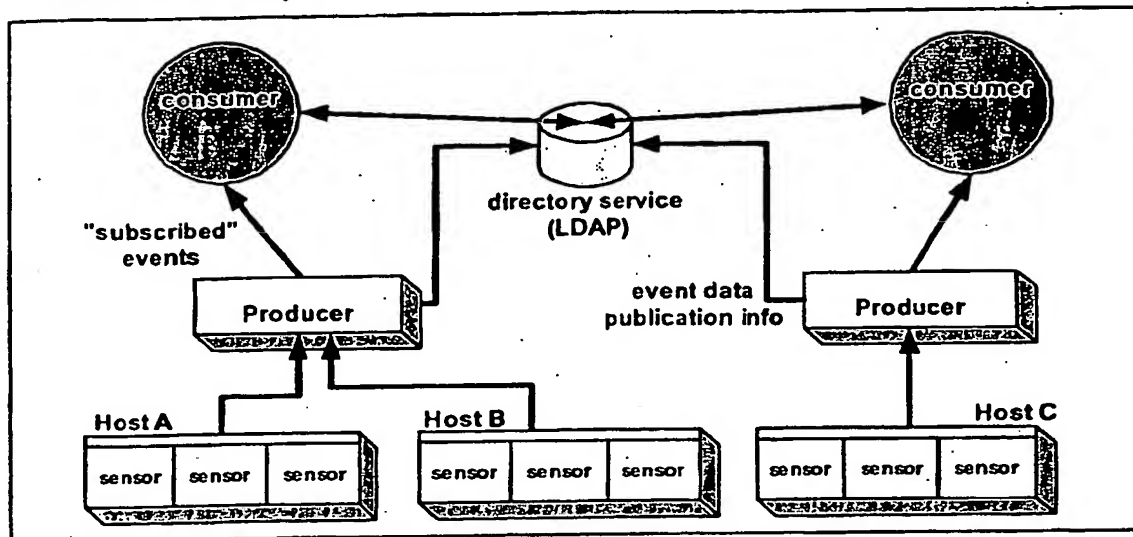


Figure 3: Relationship of Producers and Sensors

A producer may be associated with a single sensor, all sensors on a given host, all sensors on a given subnet, or any arbitrary group of sensors. This is not defined by the architecture, but is left as an implementation decision. Figure 3 shows one example of how this might be implemented. We note that there are scalability and reliability issues with how this is implemented, as described below.

Optional Producer Tasks

There are many other services that producers might provide, such as event filtering and caching. For example, producers could optionally perform any intermediate processing of the data the consumer might require. A consumer might request that a prediction model be applied to a measurement history from a particular sensor, and then be notified only if the predicted performance falls below a specified threshold. The producer might in this case filter the data for the consumer and deliver it according to the schedule the consumer determines. Another example is that a consumer might request that an event be sent only if its value crosses a certain threshold. Examples of such a threshold would be if CPU load becomes greater than 50%, or if load changes by more than 20%. The producer might also be configured to compute summary data. For example, it can compute 1, 10, and 60 minute averages of CPU usage, and make this information available to consumers. Information on which services the producer provides would be published in the directory server, along with the event information.

Protocols

The next step is to define what the protocol for consumer to producer communication, and for consumer and producer to the directory service communication. For example, current proposals include using LDAP for communicating with the directory service, and SOAP for subscribe requests. These issues will be addressed in future Global Grid Forum Performance Working Group documents.

4.0 Sample Use

An example use of the GMA is shown in Figure 4. Event data is collected on each host and at all network routers between them, and aggregated at a producer, which registers the availability of the data in the directory service. A real-time monitoring consumer subscribes to all this event data for real-time visualization and performance analysis. The producer is capable of computing summaries of network throughput and latency data, enabling a "network-aware" client [11] to optimally set its TCP buffer size. A subset of the producer's data, that from from the "server" and "router" nodes, is also sent to an archive.

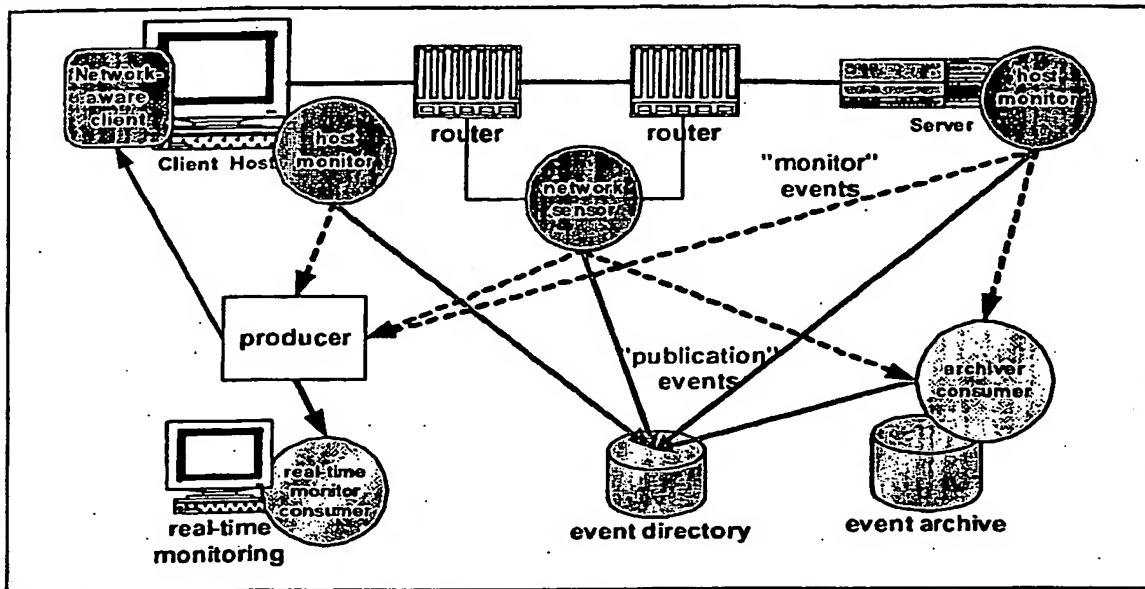


Figure 4: Sample Use of Monitoring System

5.0 Implementation Issues

The purpose of a monitoring system is to reliably deliver timely and accurate information without perturbing the system. Therefore the architecture must consider performance issues explicitly and make recommendations and requirements of implementations with the goal of avoiding services which look good on paper but which fail in practice. We discuss several of these implementation design issues below.

Monitoring service characteristics

The following characteristics distinguish performance monitoring information from other system data, such as files and databases.

Performance information has a fixed, often short lifetime of utility. Most monitoring data may go stale quickly making rapid read access important, but obviating the need for long-term storage. The notable exception to this is data that gets archived for accounting or post-mortem analysis.

- **Updates are frequent.** Unlike the more static forms of "metadata," dynamic performance information is typically updated more frequently than it is read. Most extant information-base technologies are optimized for query and not update, making them potentially unsuitable for dynamic information storage.
- **Performance information is often stochastic.** It is frequently impossible to characterize the performance of a resource or an application component using a single value. Therefore, dynamic performance information may carry *quality-of-information* metrics quantifying its accuracy, distribution, lifetime, etc., which may need to be calculated from the raw data.

- **Data gathering and delivery mechanisms must be high-performance.** Because dynamic data may grow stale quickly, the data management system must minimize the elapsed time associated with storage and retrieval. Note that this requirement differentiates the problem of dynamic data management from the problem of providing an archival performance record. The elapsed time to read an archive, while important, is often not the driving design characteristic for the archival system. We believe that archival data is useful both for accounting purposes and for long-term trend analysis. It is our belief; however, the separate but complimentary systems for managing and archiving Grid performance data respectively are required, each tailored to meet its own set of unique performance constraints.
- **Performance measurement impact must be minimized.** There must be a way for monitoring facilities to be able to limit their intrusiveness to an acceptable fraction of the available resources. If no mechanism for managing performance monitors is provided, performance measurements may simply measure the load introduced by other performance monitors.

General Implementation Strategies

A number of the authors of this white paper have built various monitoring systems. The following lessons have been learned from this experience, and should be considered when implementing a monitoring system.

- **The data management system must adapt to changing performance conditions dynamically.** Dynamic performance data is often used to determine whether the shared Grid resources are performing well (e.g. fault diagnosis) or whether Grid load will admit a particular application (e.g. resource allocation and scheduling). To make an assessment of dynamic performance fluctuation available, the data management system cannot, itself, be rendered inoperable or inaccessible by the very fluctuations it seeks to capture. As such, the data management system must use the data it gathers to control its own execution and resources in the face of dynamically changing conditions.
- **Dynamic data cannot be managed under centralized control.** Having a single, centralized repository for dynamic data (however short its lifespan) causes two distinct performance problems. The first is that the centralized repository for information and/or control represents a single-point-of-failure for the entire system. If the monitoring system is to be used to detect network failure, and a network failure isolates a centralized controller from separate system components, it will be unable to fulfill its role. All components must be able to function when temporarily disconnected or unreachable due to network or host failure. For example, a producer must still be able to accept connections from consumers even if its connection to sensors or the directory server is down. In addition, once access is restored, producers must be able to reconfigure themselves automatically with respect to the rest of the running service components. A second problem with centralized data management is that it forms a performance bottleneck. For dynamic data, writes often outnumber reads. That is, performance data may be gathered that is never read or accessed since demand for the data cannot be predicted. Experience has shown that a centralized data repository simply cannot handle the load generated by actively monitored resources at Grid scales.
- **All system components must be able to control their intrusiveness on the resources they monitor.** Different resources experience varying amounts of sensitivity to the load introduced by monitoring. A two megabyte disk footprint may be insignificant within a 10 terabyte storage system, but extremely significant if implemented for a palm-top or RAM disk. In general, performance monitors and other system components must have tunable CPU, communication, memory, and storage requirements.
- **Efficient data formats are critical.** In choosing a data format, there are trade offs between ease-of-use and compactness. While the easiest and most portable format may be ASCII text including both event item descriptions and event item data in each transmission, this also the least compact. This format may be suitable for cases where a small amount of data is recorded and transmitted infrequently. However, some sources of event data can generate huge volumes of data in a short amount of time, demanding that a more efficient data format be adopted. Compressed binary representations that can be read on machines with different byte orders is one possibility. Transmitting only the item data values and using a data structure obtained separately to interpret the data is another way to reduce the data volume. XML is an emerging standard that allows the data description to be separated from the data values. The XML schema could be placed in a separate directory server, retrieved, and used in conjunction with the

event data values. Another possibility is to send the data descriptor one time when a consumer subscribes to a producer, and send only the data values for each event transmission. The GMA could support registration of a data format for each event, allowing different events to use the format most appropriate for their needs. Consumers could be provided plug-in modules to convert from one format to another.

Scalability

One of the biggest issues in defining a monitoring architecture for use in a Grid environment is scalability. It is critical that the act of monitoring has minimal affect on the systems being monitored. In this model, one can add additional producers and additional directory servers as needed, reducing the load where necessary. In the case where many consumers are requesting the same event data, the use of a producer reduces the amount of work on and the amount of network traffic from the host being monitored. As such, the resources that a producer will use must, themselves, be scheduled. A producer might be run on a separate host from the Grid resources, to ensure that the load from the producer did not affect what was being monitored.

In particular, we believe that the GMA is more scalable than the CORBA Event Service. In the GMA, event data is not sent anywhere unless it is requested by a consumer. Many of the current event service systems, including CORBA, send all event data to a central component, which consumers then contact. In the GMA, only event data subscription information (i.e.: which producer to contact) is sent to a central directory server. Event data goes directly from producer to consumer. We believe this model will scale much better in a Grid environment.

In addition, for the GMA system to scale, performance monitoring consumers (particularly those that require the cooperation between two or more producers) must coordinate their interactions to control intrusiveness. For example, if network performance is to be monitored between all pairs of hosts attached to a single Ethernet segment, the network probes required to generate end-to-end measurements cannot occur simultaneously. If they do, both the quality of the readings that are gathered and the network capacity that is available for other work will suffer. If performance monitors are not coordinated in the Grid, the intrusiveness of performance monitoring may strongly impact available performance, particularly as the system scales. That is, if all performance facilities operate their own monitoring sensors, Grid resources will be consumed by the monitoring facilities alone. Coordinating a Grid-wide collection of sensors is complicated both by the scale of the problem (there are many Grid resource characteristics to monitor) and by the dynamically changing performance and availability of Grid resources that are being used to implement the dynamic data management service.

One recommended producer service that is important for system scalability is that of consumer-specified caching. Often a consumer needs to access only a small subset of the global data pool, and will sacrifice fast access for tight data consistency. An automatic program scheduler, for example, might want the "freshest" data that can be delivered for a specified set of hosts with no more than a one second access delay. To achieve this functionality at Grid scales, producers must cache the data the consumer will want and deliver whatever data is available at the time of request. Experience with dynamic program scheduling indicates that this type of producer is valuable to scalable performance within the Grid [2].

Security Issues

A distributed system such as this creates a number of security vulnerabilities which must be analyzed and addressed before such a system can be safely deployed on a production Grid. The users of such a system are likely to be remote from the machines being monitored and to belong to different organizations.

Typical user actions will include queries to the directory service concerning event data availability, subscriptions to producers to receive event data, and requests to instantiate new event monitors or to adjust collection parameters on existing monitors. In each case, the domain that is being monitored is likely to want to control which users may perform which actions.

Public key based X.509 identity certificates [6] are a recognized solution for cross-realm identification of users. When the certificate is presented through a secure protocol such as SSL (Secure Socket Layer), the server side can be assured that the connection is indeed to the legitimate user named in the certificate.

User (consumer) access at each of the points mentioned above (directory lookup and requests to a producer), would require an identity certificate passed through a secure protocol, e.g. SSL. A wrapper to the directory server and the producer could both call the same authorization interface with the user's identity and the name of the resource the user wants to access. This authorization interface could return a list of allowed actions, or simply deny access if the user is unauthorized. Communication between the producer and the sensors may also need to be controlled, so that a malicious user can not communicate directly with the monitoring process.

6.0 Related Work

There are many existing systems with an event model similar to the one described here. CORBA includes an "event service" [1] that has a rich set of features, including the ability to push or pull events, and the ability for the consumer to pass a filter to the event supplier. JINI also has a "Distributed Event Specification" [7], which is a simple specification for how an object in one Java™ virtual machine (JVM) registers interest in the occurrence an event occurring in an object in some other JVM, and then receives a notification when that event occurs. There are also several other systems with alternative event models, such as the Common Component Architecture; many of which are summarized in [8]. However, we believe that none of the existing systems is a perfect match for a Grid monitoring system; therefore we have tried to combine the relevant strengths of each. Another related system is Autopilot [9], which has had the notion of *sensors* for several years, and which implements a similar publish/lookup/subscribe architecture. Note that this list of systems is not intended to be exhaustive, but only illustrative of the usefulness of the proposed architecture.

7.0 Acknowledgements

Input from many people went into this document, including almost all attendees of the various Grid Forum meetings. The LBNL portion of this paper was supported by the U. S. Dept. of Energy, Office of Science, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division, under contract DE-AC03-76SF00098 with the University of California.

8.0 References

- [1] CORBA, "Systems Management: Event Management Service", X/Open Document Number: P437, <http://www.opengroup.org/onlinepubs/008356299/>
- [2] Dail, H, G. Obertelli, F. Berman, R. Wolski, and A. Grimshaw "Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem", Proceedings of the 9th Heterogeneous Computing Workshop, May 2000.
- [3] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations". In *Proceedings 6th IEEE Symposium on High Performance Distributed Computing, August 1997*.
- [4] The Globus project: See <http://www.globus.org>
- [5] The Grid: Blueprint for a New Computing Infrastructure, edited by Ian Foster and Carl Kesselman. Morgan Kaufmann, Pub. August 1998. ISBN 1-55860-475-8.
- [6] Housely, R., W. Ford, W. Polk, D. Solo, "Internet X.509 Public Key Infrastructure", IETF RFC 2459. Jan. 1999
- [7] Jini Distributed Event Specification", <http://www.sun.com/jini/specs/>
- [8] Peng, X, "Survey on Event Service", <http://www-unix.mcs.anl.gov/~peng/survey.html>
- [9] Randy L. Ribler, Jeffrey S. Vetter, Huseyin Simitci, and Daniel A. Reed, "Autopilot: Adaptive Control of Distributed Applications," Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.
- [10] W. Smith, "Monitoring and Fault Management," http://www.nas.nasa.gov/~wwsmith/mon_fm
- [11] Tierney, B., B. Crowley, D. Gunter, M. Holding, J. Lee, M. Thompson A Monitoring Sensor Management System for Grid Environments Proceedings of the IEEE High Performance Distributed Computing conference (HPDC-9), August 2000, LBNL-45260.
- [12] Tierney, B. Lee, J., Crowley, B., Holding, M., Hylton, J., Drake, F., "A Network-Aware Distributed Storage Cache for Data Intensive Environments", Proceeding of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896. <http://www-didc.lbl.gov/DPSS/>
- [13] Tierney, B., W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter, "The NetLogger Methodology for High Performance Distributed Systems Performance Analysis", Proceeding of IEEE High Performance Distributed Computing conference, July 1998, LBNL-42611. <http://www-didc.lbl.gov/NetLogger/>
- [14] A. Waheed, W. Smith, J. George, J. Yan. "An Infrastructure for Monitoring and Management in Computational Grids." In *Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems*
- [15] Wahl M., Howes, T., Kille S., "Lightweight Directory Access Protocol (v3)", Available from <ftp://ftp.isi.edu/in-notes/rfc2251.txt>
- [16] Wolski, R., Spring, N., Hayes, J., "The Network Weather Services: A Distributed Resource Performance Forecasting Service for Metacomputing," Future Generation Computing Systems, 1999. <http://nsw.npaci.edu/>

3LS - A Peer-to-Peer Network Simulator

Nyik San Ting, Ralph Deters
 Department of Computer Science
 University of Saskatchewan
 Saskatoon, Saskatchewan,
 S7N 5A9 Canada.
 nyt431@mail.usask.ca
 ,deters@cs.usask.ca

Abstract

Peer-to-Peer (p2p) networks are the latest addition to the already large distributed systems family. With a strong emphasis on self-organization, decentralization and autonomy of the participating nodes, p2p-networks tend to be more scalable, robust and adaptive than other forms of distributed systems. The much-publicized success of p2p-networks for file-sharing and cycle-sharing have resulted in an increased awareness and interest into the p2p protocols and applications. However, p2p-networks are difficult to study due to their size and the complex interdependencies between users, application, protocol and network. This paper presents a 3-level simulator designed to study complex p2p networks.

1. P2P-Network Simulation

The field of P2P networks is still undergoing major changes with new applications and protocols emerging on a nearly monthly basis. However, due to the difficulties in evaluating them prior to their large-scale deployments, they are often short-lived – disappearing as fast as they emerge – normally due to bad performance. What seemed to work well when using a small number of nodes, high bandwidth, low latency, attractive services/content and highly cooperative users often fails in real world deployments.

Testing a system performance prior to its deployment is a fairly common element in the software development of applications.

P2P networks tend to be large, heterogeneous systems with complex interactions between the physical machines, underlying network, application and user. Hence, testing of a “running” p2p-network or protocol in a realistic environment is often not feasible. However, it is possible to use a simulation of a p2p-network to evaluate the applications and protocols in controlled environment.

Researchers, who wanted to simulate a p2p system, tend to avoid the development of a complex simulator and focus on some selected areas (such as caching schemes). While

some may choose to start an implementation from scratch an increasing number of researchers build their simulators on top of existing tools (e.g. the agent platform JADE [2]) to speed-up the development. The general problem of having only special-purpose simulators is that the results obtained with one simulator are difficult to validate and often impossible to achieve with another simulator due to the many hard-coded assumptions of every simulator.

Figure 1 shows a high level view of the 3LS simulator. 3LS is a time-stepped simulator that uses a central step-clock is used to simulate the timing. In 3LS the models for network, p2p protocol and user model are clearly separated. With the separation of the network, protocol and application model from each other, the simulation of various network topologies, for different protocol, applications, and user models becomes possible. Hence, three levels have been defined:

- Network level (bottom),
- Protocol level (middle) and
- User level (top).

Communication can only happen between the directly connected levels. The protocol-level, that is responsible for simulating the p2p-protocol and application, and serves as the interface between the user-level and the network-level. Input information from the user is fed into the network level through a GUI interface or a file. Upon starting the simulator it is possible to either create the models (fig. 2) for the three described levels or to choose among a library the ones most suited models/combination for the simulation run. As the simulation is running, the events are displayed on the command prompt screen. After the simulation has been completed, all simulation data is saved into a file for future analysis. Though simulation languages provide most of the features needed in programming a simulation model and the details of the simulation models can be easily changed, a general-purpose language was selected to provide “greater programming flexibility”. Since Java is the preferred language of many p2p programmers it was chosen as the host-language for the 3LS simulator. Visualization of the network is done with the aid of the tool

AiSee [1]. AiSee was selected for its, ease of use, simple installation, availability (runs under various OS), functionality and performance in rendering. When screenshots of the p2p-network are to be visualized, files containing the information of the graph are created by 3LS using the Graph Description Language (GDL). Once the file is created a user can use AiSee to render an image of the graph (see figure 3).

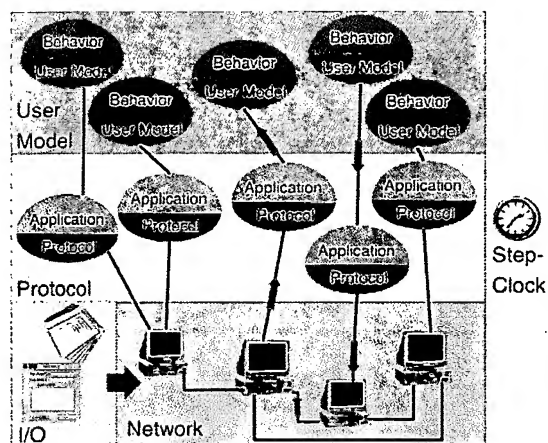


Figure 1: Architecture of the 3LS

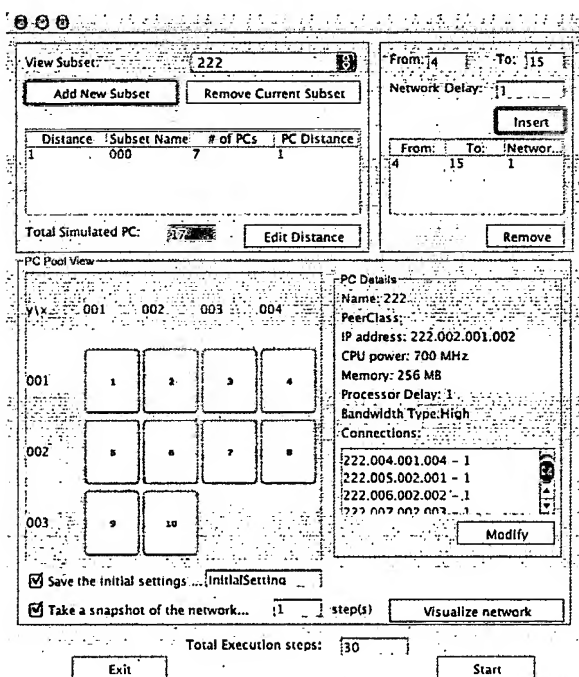


Figure 2: Screenshot of 3LS - Network-Model

2. Future Work

Future work focuses on collecting data for the various layers e.g. human desktop usage and network traffic. We

are currently testing the simulator by comparing its results for a Gnutella 0.4 network [3] with the "real data" obtained from running Gnutella 0.4 clients in a controlled network. Using Comella [4] clients we are able to adjust the various parameters of the simulation and verify the simulation results. Early results in a small network (less than 20 nodes) indicate that the simulator works as expected but more testing is needed.

3. Code

The complete code of the 3LS simulator is available upon request by sending an email to one of the authors. 3LS requires a Java 1.3.1 or higher version of the JDK.

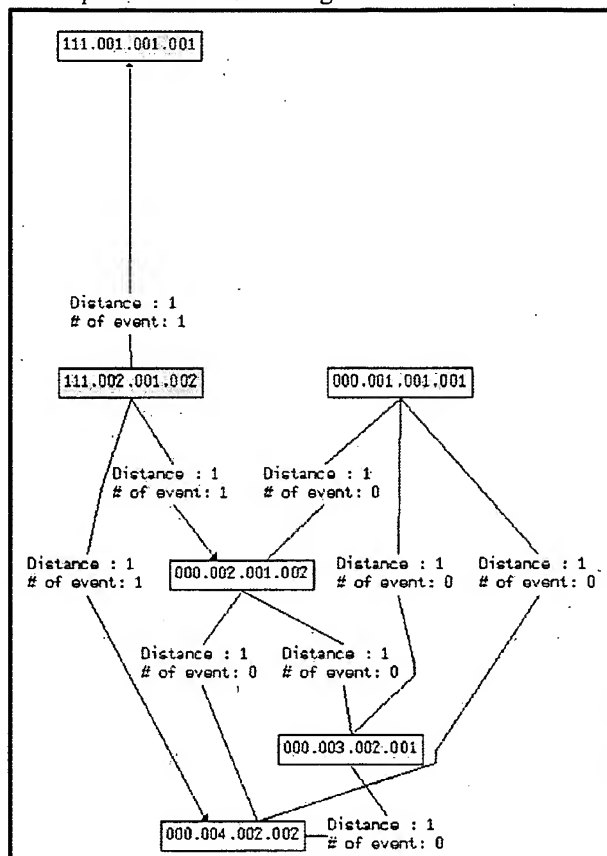


Figure 3: Example of network view using AiSee.

4. REFERENCES

- [1] AbsInt. AiSee homepage. <http://www.aisee.com/>
- [2] Bellifemine, F., Poggi, A., Rimassa, G. JADE-A FIPA-compliant agent framework In Proceedings of PAAM'99, London, April 1999, 97-108.
- [3] Clip2. The Gnutella Protocol Specification v0.4. http://rfcgnutella.sourceforge.net/Development/GnutellaProtocol0_4-rev1_2.pdf
- [4] Vassileva, J. Motivating participation in Peer-to-Peer Communities. Proceeding of Workshop on Emergent

Societies in the Agent World, ESAW'02, Madrid, 16-17 September, 2002.

Grid Organization

CROSS-REFERENCE TO RELATED APPLICATIONS

This application incorporates by reference the content of U.S. Provisional Application No. 60/490,818, Express Mail Number, EV 331001684 US, filed July 28, 2003, to Erol Bozak *et al.*, entitled GRID COMPUTING MANAGEMENT.

5

TECHNICAL FIELD

The present invention relates to data processing by digital computer, and more particularly to a dynamic tree structure for grid computing.

BACKGROUND

10

In today's data centers, the clusters of servers in a client-server network that run business applications often do a poor job of managing unpredictable workloads. One server may sit idle, while another is constrained. This leads to a "Catch-22" where companies, needing to avoid network bottlenecks and safeguard connectivity with customers, business partners and employees, often plan for the highest spikes in workload demand, then watch as those surplus servers operate well under capacity most of the time.

15

In grid computing, all of the disparate computers and systems in an organization or among organizations become one large, integrated computing system. That single integrated system can then handle problems and processes too large and intensive for any single computer to easily handle in an efficient manner.

20

More specifically, grid computing is a form of distributed system wherein computing resources are shared across networks. Grid computing enables the selection, aggregation, and sharing of information resources resident in multiple administrative domains and across geographic areas. These information resources are shared, for example, based upon their availability, capability, and cost, as well as a user's quality of service (QoS) requirements. Grid computing can mean reduced cost of ownership, aggregated and improved efficiency of

25

computing, data, and storage resources, and enablement of virtual organizations for applications and data sharing.

SUMMARY

In one aspect, the invention features a method that includes, in a client server network, maintaining systems having grid managers having hierarchical relations, the relations of each grid manager stored in each of the systems.

5 Embodiments may include the following. Each of the hierarchical relations are classified as superior or inferior.

In another aspect, the invention features a system that includes a network of computer systems, each of the computer systems including a grid management engine, each of the grid managers having hierarchical relations with other grid managers, the relations of each grid
10 manager stored in each of the systems.

Embodiments may include the following. Each of the relations are classified as superior or inferior.

In another aspect, the invention features a method that includes, in a network, starting an execution of a first service on a first computer, the first service handling at least locating,
15 reserving, allocating, monitoring, and deallocating one or more computational resources for one or more applications using the network. The method further includes reading, by the first service, a file to inform the first service of a relation with a second service, wherein the first service has a inferior relation with the second service, the inferior relation meaning that the second service can send a query for available computer resources to the first service. The
20 method further includes establishing a first communication channel from the first service to the second service, and accepting an opening of a second communication channel from the second service to the first service.

Embodiments may include one or more of the following. The method includes receiving a message to cancel the first service's inferior relation with the second service, closing the first
25 and second communication channels, receiving a message to generate a inferior relation from the first service to a third service residing in a third computer, establishing a third communication channel from the second service to the third service, and accepting an opening of a fourth communication channel from the third service to the first service. In some cases, establishing a first communication channel further includes determining if the second service responds to
30 determining and if not, establishing a communication channel to the second service after a predetermined time period.

In another aspect, the invention features a method that includes, in a network, starting an execution of a first service residing in a first computer, the first service handling at least locating, allocating, monitoring, and deallocating one or more computational resources for one or more applications using the network, starting an execution of a second service residing in a second computer, and reading, by the second service, a file to inform the second service of a relation with the first service, wherein the second service has a inferior relation with the first service, wherein the inferior relation indicates that the first service can send a query for available computer resources to the second service. The method further includes establishing a first communication channel from the second service to the first service, and establishing a second communication channel from the first service to the second service.

Embodiments may include the following. The method of further includes receiving, by the second service, a message to cancel the second service's relation with the first service, closing the first communication channel, failing to respond to the second communication channel, receiving a message to create a inferior relation from the second service to a third service, establishing a third communication channel from the second service to the third service, and establishing a fourth communication channel from the second service to the third service.

In another aspect, the invention features a system that includes two or more computers each configured to run a service, the service handling at least locating, allocating, monitoring, and deallocating one or more computational resources for one or more applications. The system also includes a network of the services, the network configured such that a first service from the services has a superior relation with a second service from the services and the second service has an inferior relation with the first service, wherein the first service is configured to check the status of the second service in the network by waiting for a response to a query from the first service to the second service.

Embodiments may include the following. The relation includes a first communication channel from the first service to the second service and a second communication channel from the second service to the first service. The first service is further configured to locate the one or more computational resources for the one or more applications by sending a query for available computational resources to the second service. The second service is further configured to remove its inferior relation with the first service and create a new superior relation with a third service.

These and other embodiments may have the following advantage. A fast and robust grid computing environment can be achieved using the dynamic tree structure for grid management.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram of a grid computing environment.

FIG. 2 is a flow diagram for discovering and reserving resources in the grid computing environment of FIG. 1.

FIG. 3 is a flow diagram for installing, running, and removing applications in the grid computing environment of FIG. 1.

FIG. 4 is a block diagram of a computer device in the grid computing environment of FIG. 1.

FIG. 4A is a flow diagram for starting up an application in the computer device of FIG. 4.

FIG. 5 is a flow diagram for starting up grid managers in the grid computing environment of FIG. 1.

FIG. 5A is a block diagram of the grid computing environment of FIG. 1 that is augmented with another computer device.

FIG. 6 is a block diagram of an exemplary a grid graphical user interface (GUI) component for visualization of a grid computing environment.

FIG. 7 is a block diagram of a grid browser component.

Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

As shown in FIG. 1, services in a grid computing environment **100** manage computational resources for applications. The grid computing environment **100** is a set of distributed computing resources that can individually be assigned to perform computing or data retrieval tasks for the applications. The computational resources include computer devices **12**, **14**, **16**, **18**, **20**, and **22**. The computer devices communicate using a network **8**. The applications have scalable computational requirements. For example, an example application that uses computer devices **12**, **14**, **16**, **18**, **20**, and **22** in the grid computing environment **100** is an internet

pricing configurator. The computer device 12 provides network access to pricing information to users via web browsers on computer devices that are connected to the internet. The web browsers can be any application able to display content and/or execute applications such as web pages, media files, and programs, such as Netscape Navigator®, Microsoft Internet Explorer®, and similar applications.

In this example, a web server on computer device 12 provides pricing information to the users. Calculation parameters for each price to be calculated are passed by an IPC dispatcher 116 to IPC servers 120, 122, 124, and 126 that execute on computer devices 12, 14, 16, and 18, respectively. Due to the flexibility of the web server and applications on the internet, the number of users can vary. This generates dynamic computational requirements for the internet pricing configurator. An IPC manager 118 communicates with services in the grid computing environment 100 so that the services can allocate and deallocate computational resources (e.g., processors in computer devices 12, 14, 16, 18, 20, 22) based on the dynamic computational requirements of the internet pricing configurator. Allocating and deallocating computational resources in this manner allows computer devices 12, 14, 16, 18, 20, or 22 to be designated as general-purpose computational resources and not solely dedicated to handling peak demands of the internet pricing configurator application. The IPC manager 118 coordinates with the IPC dispatcher 116 so that the IPC dispatcher 116 has access to resources in network 8.

This capability to allocate and deallocate the resources in the grid computing environment 100 enables the IPC manager 118 to locate and use available computational resources on an "as needed" basis. Once resources are located, the IPC manager 118 can use services in the grid computing environment 100 to install the IPC servers 120, 122, 124, and 126 as applications on computer devices in the grid computing environment 100. The IPC dispatcher 116 uses Web Service Definition Language (WSDL) interfaces defined in the Open Grid Services Infrastructure (OGSI) Version 1.0 by Tuecke *et al* to manage and exchange the information flow between the IPC dispatcher 116 and IPC servers 120, 122, 124, and 126. For example, the OGSI WSDL interfaces can be used to pass computation parameters for pricing calculations from the IPC dispatcher 116 and the IPC servers 120, 122, 124, and 126. The OGSI WSDL interfaces can also be used to pass completed results from the IPC servers 120, 122, 124, and 126 back to IPC dispatcher 116. The OGSI Version 1.0 is incorporated herein by reference.

The OGSi WSDL interfaces enable the controlled, fault-resilient, and secure management of the grid computing environment 100 and applications such as the internet pricing configurator.

While the IPC dispatcher 116 uses IPC servers 120, 122, 124, and 126 to perform calculations for users, services in the grid computing environment 100 monitor resource utilization on computer devices in the grid computing environment 100 running the IPC servers 120, 122, 124, and 126. The services also send this utilization information to the IPC manager 118. Based on a comparison between utilization requirements and current resource loading, the IPC manager 118 can dynamically inform services in the grid computing environment 100 to allocate more resources for IPC servers 120, 122, 124, and 126 or deallocate resources to keep utilization of resources in the grid computing environment 100 at a desired level.

Grid managers 152, 154, 156, 160, 162, and 164 are resident in computer devices 12, 14, 16, 18, 20, and 22, respectively. Within the grid computing environment 100, pairs of grid managers can have directional relations that classify one grid manager as superior to another grid manager. A grid manager can have more than one superior relations with other grid managers. For example, grid manager 152 has a superior relation with grid managers 154 and 156. A grid manager can also have more than one inferior relations with other grid managers. Through these hierarchical relations, IPC manager 118 does not need access to a list of all computer devices in network 8 to use the computational resources in the grid computing environment 100. IPC manager 118 is only required to have access to a network address of one computer device running a grid manager (e.g., computer device 12 running grid manager 152) and this grid manager uses its relations with other grid managers running on other computer devices to provide IPC dispatcher 116 with indirect access to other computer devices in the grid computing environment 100.

A grid manager (e.g., 152, 154, 156, 160, 162, and 164) maintains a first list of all superior relations with other grid managers and a second list of all inferior relations with other grid managers. Each grid manager maintains an "always open" communications channel to all the grid managers in these lists over network 8 using, for example, the aforementioned OGSi WSDL interfaces on transmission control protocol (TCP), hypertext transfer protocol (HTTP), and simple object access protocol (SOAP). These lists and corresponding communication channels can be modified, allowing a dynamic reconfiguration of the grid hierarchy during runtime. This also allows a failing grid manager to be dynamically replaced in the hierarchy.

For example, referring to FIG. 1, if grid manager 154 fails, then grid manager 152 loses its connection to grid managers 160 and 162. In this case, relations between grid managers can be modified so that grid manager 152 has new superior relations to grid managers 160 and 162. Likewise, grid managers 160 and 162 have new inferior relations to grid manager 152.

5 As shown in FIG. 2, an application start process 200 is designed so applications (e.g., internet pricing configurator) get necessary resources allocated in the network 8 before executing on a computer device (e.g., 12, 14, 16, 18, 20, or 22). Process 200 also guarantees if similar applications are trying to start at the same time on the same resource on a computer device that the two or more applications do not collide or interfere with each other. For example, the IPC
10 manager 118 can require that an IPC server (e.g., 120) be the only application executing on a processor in computer device 14 for quality of service (QoS). In this case, another application would interfere if the other application simultaneously attempted to execute on the processor in computer device 14.

Process 200 includes IPC manager 118 (or some other application) sending (202)
15 requirements for computational resources to query a grid manager (e.g., 154) to determine if there are resources matching these requirements available in the grid computing environment 100. These requirements specify information pertaining to resources in a computer device such as required number of processors, required percentage of utilization for those processors, main memory, and network speed. The query can also include information to which hierarchy level
20 (in the grid computing environment 100) the query should be propagated. Process 200 includes grid manager 154 receiving (204) the requirements.

To respond to the query for available resources from IPC manager 118, process 200 includes grid manager 154 matching (206) the requirements against resources known to grid manager 154. These resources include resources (e.g., processor 40) in computer device 14 that
25 are directly managed by grid manager 154. Resources directly managed by grid manager 154 that are currently available and meet the requirements are added to a resource-query list maintained by grid manager 154.

Grid manager 154 also sends the query to grid managers 160 and 162 having inferior relations with grid manager 154. Process 200 includes grid managers 160 and 162 responding
30 (208) to the query by sending to grid manager 154 lists of resources (e.g., processors on computer devices 18, 20) that meet the requested requirements and are available and known to

grid managers 160 and 162, respectively. These resource-query lists of resources that are known to grid managers 160 and 162 can also include resources managed by grid managers (not shown) with inferior relations to grid managers 160 and 162. Grid manager 154 adds these resource-query lists of available resources from grid managers 160 and 162 to its resource-query list of available resources meeting the requested requirements. If process 200 determines (210) that there is at least one resource (e.g., processor 40) in this resource-query list, then grid manager 154 sends (214) this resource-query list to IPC manager 118. Otherwise, if process 200 determines (212) that grid manager 154 has a relation with a superior grid manager (e.g., grid manager 152), grid manager 154 sends (202) the query for available resources to grid manager 152. In response to this query, grid manager 152 does not send a redundant query back to grid manager 154 having an inferior relation with grid manager 152.

Process 200 includes grid manager 154 sending (214) the list of available resources along with addresses of their corresponding grid managers in the network 8 that match the requirements. The IPC manager 118 selects a resource (e.g., on computer device 16) from the list and requests (216) a reservation of the resource on computer device 16 to the grid manager 154 managing the resource on computer device 16. If the resource in computer device 16 is still available for reservation (218) and the reservation succeeds, grid manager 154 sends (220) a reservation number to the IPC manager 118. This reservation means that the IPC manager 118 is guaranteed and allocated the requested resource on the computer device 16 in the grid computing environment 100. The grid manager 154 handles queries for available resources from applications such as IPC manager 118 using independent processing threads of execution. Thus, the grid manager 154 uses a semaphore to ensure that the same resource (e.g., processor 40) is not assigned multiple reservation numbers for different applications simultaneously requesting the same resource.

If the grid manager determines that the requested resource in computer device 16 is not available for reservation and the reservation fails, the IPC manager 118 selects the next available resource in the list and requests (216) the reservation of this next available resource. If the IPC manager 118 receives a registration number and a timeout measured from the sending of the registration number does not expire (222), the IPC manager 118 starts (224) the IPC server 122 on the processor 40 resource in computer device 16. Starting the IPC server 122 is initiated by passing the reservation number and an application file to the grid manager 156 and then grid

manager 156 reads the application file to install and execute the IPC server 122 on computer device 16.

As shown in FIG. 3, process 250 installs an application (e.g., IPC server 122) on a computer device (e.g., 14) in the grid computing environment 100 to set up an available resource for the application, using the available resource, and removing or deinstalling the application to free up the resource for use by subsequent applications when the resource is no longer needed. Process 250 includes IPC manager 118 transferring (252) an application file containing code for IPC server 122 in addition to instructions on how to install, customize, track and remove the application from computer device 14 so that the grid manager 154 can return computer device 14 to an original state after executing the application.

IPC manager 118 transfers the application file using a file transfer protocol (FTP), hypertext transfer protocol (HTTP), or a file copy from a network attached storage (NAS) for example, to computer device 14 as a single file, such as a compressed zip file. Within this zip file there is information about installing and customizing the application IPC server 122. This information is represented by a small executable program or extended markup language (XML) document that is extracted and interpreted (254) by an installation and customizing engine (not shown) in grid manager 154. Process 250 includes grid manager 154 installing (256) and running (258) the application. During installation (256), customization and execution (258) of the application, all changes to the computer device 14 are logged so that when the application is terminated (260) or deinstalled by grid manager 154 upon request by IPC manager 118, grid manager 154 removes the application from the computer device 14 and also removes (262) any other changes to computer device 14 that were done when installing and running the application. Thus, the computer device 14 reverts to its original state prior to execution of the application and all of the resources of computer device 14 are again available for use by a subsequent application. This allows the resources to become available after running the application without rebooting computer device 14. These changes include space in memory (e.g., 32) allocated to store and run application code in addition to other changes such as allocation of communication ports.

In some examples, multiple applications can simultaneously run on resources in a single computer device (e.g., 14). Applications for the grid computing environment 100 are classified in part based on their resource requirements. Some changes to a computer device to run an

application are only required for the first execution of an application of its class and subsequent executions do not require these changes. In these examples, grid manager 154 only does the changes for the first execution. Furthermore, when deinstalling the applications, grid manager 154 only removes the changes for the last application that was executed and terminated.

5 After installing applications on computer devices in the grid computing environment 100, grid managers are configured to start or stop the processes of these applications upon request. In the example of the internet pricing configurator (IPC) application, grid manager 154 is configured to start or stop IPC server 122 on computer device 14 after installing IPC server 122 on computer device 14. The IPC manager 118 requests grid managers to start or stop IPC
10 servers in the grid computing environment 100 based on current utilization of resources in the grid computing environment 100. After stopping IPC server 122 on computer device 14, IPC manager 118 waits a prespecified amount of time and then requests grid manager 154 to deinstall IPC server 122 if current resource utilization does not indicate a need to start IPC server 122 again. Furthermore, as mentioned previously, grid managers monitor resource utilization on
15 computer devices such as computer device 14 running applications (e.g. IPC servers 120, 122, 124, and 126) and send this utilization information to IPC manager 118.

In many examples, control of application processes on resources in a computer device is specific to the operating system (OS). The grid computing environment 100 is configured to handle different operating systems on computer devices. Furthermore, grid computing
20 environment 100 is designed to handle different applications (e.g., internet pricing configurator) that do not have to be redesigned to execute on the grid computing environment 100. A grid manager controls an application process in a general manner that decreases interdependence between development of grid manager code and application code. An interface is provided to application code to enable grid managers to discover, control (e.g., start, stop, halt, resume) and
25 inspect or monitor a state of application processes. The interface is provided for operating system processes that are exposed by the operating system or hosting environment and includes three aspects. One aspect of the interface is process data, such as process identification, states, degree of resource consumption (such as Central Processing Unit (CPU), memory, socket bindings, or other resources that an application can use), and application specific data defined by
30 a process data scheme.

A second aspect of the interface is managing operations, such as start, stop, wait, resume, change priority, and other operations defined by supported managing operations.

A third aspect of the interface is control bindings and definitions, such as process data scheme, supported managing operations, and communication bindings. Since not all applications
 5 running in the grid computing environment 100 have access to the same information and capabilities in these three aspects, the applications provide to grid managers a list of queries and commands that each application supports.

The interface provided to application code is an Application Program Interface (API). The API is a set of methods (embedded in software code) prescribed by the grid manager
 10 software by which a programmer writing an application program (e.g., internet pricing configurator) can handle requests from the grid manager.

As shown in FIG. 4, IPC server 122 includes an API 302 and a document 304. Since the API 302 is adapted to different types of applications, the document 304 describes how grid
 manager 154 communicates with the IPC server 122 and what requests through the API 302 are
 15 supported by the IPC server 122. Grid manager 154 reads document 304 before starting up IPC server 122. In some examples, document 304 is written in XML and includes a Document Type Description (DTD) 306. A DTD is a specific definition that follows the rules of the Standard
 Generalized Markup Language (SGML). A DTD is a specification that accompanies a document and identifies what the markups are that separate paragraphs, identify topic headings, and how
 20 each markup is to be processed. By including the DTD 306 with document 304, grid manager 154 having a DTD "reader" (or "SGML compiler") is able to process the document 304 and can correctly interpret many different kinds of documents 304 that use a range of different markup codes and related meanings.

As shown in FIG. 4A, grid manager 154 uses process 350 to install applications such as
 25 IPC server 122. Grid manager 154 reads (352) DTD 306 in document 304 to identify markups in document 304. Grid manager 154 reads (354) document 304 using markups to identify communication parameters for communicating with IPC server 122. Grid manager 154 sets up
 (356) communications with IPC server 122 based on the specifications of the communication parameters. Grid manager 154 communicates (358) with IPC server 122 using the
 30 communication parameters to send requests such as "Start", "Stop", and "Are you idle?".

Before any applications (e.g., internet pricing configurator) can be executed on network 8, grid managers 152, 154, 156, 160, 162, and 164 are asynchronously started up on computer devices 12, 14, 16, 18, 20, and 22, and relations to other grid managers are established. As shown in FIG. 5, process 400 initializes relations among grid managers. For each grid manager (e.g., grid manager 154), the grid manager 154 starts up on computer device 14 by reading (402) a properties file. The properties file contains a list of addresses of computer devices with grid managers having superior relations to grid manager 154. This list was described earlier as a first list of all superior relations with other grid managers. If (404) a superior grid manager (e.g., grid manager 152) is specified in this list of addresses, grid manager 154 requests (406) to open a communication channel to the superior grid manager (e.g., 152). If grid manager 152 is already started, then grid manager 152 responds by accepting the request of the opening of the communication channel from grid manager 152. Process 400 includes grid manager 154 detecting (408) any requests for communication channels from grid managers (e.g., grid managers 160, 162) identified as having inferior relations with grid manager 154. If process 400 determines (410) that there are some requests, grid manager 154 allows communication channels from the inferior grid managers (e.g., 160, 162). Process 400 includes grid manager 154 checking (414) if there are any pending requests for communication to grid managers having superior relations. If there are any pending requests, grid manager 154 requests (406) communication channels to grid managers. These communication channels are used for resource queries between grid managers (as described previously) and "heart beat" messages between grid managers to ensure that each grid manager in the grid computing environment 100 is functioning.

Once grid managers 152, 154, 156, 160, 162, and 164 are running with established relations, the grid managers are used for the proper operation of the grid computing environment 100. Often during the lifecycle of the grid computing environment 100 the functionality of the grid managers are enhanced. It is often not possible or convenient to shut down the grid computing environment 100 and start the grid computing environment 100 up with the enhancements. Grid managers 152, 154, 156, 160, 162, and 164 are configured so that there is only a minimal impact on users of the grid computing environment 100 when a change happens. To enable this transparency, an API is provided for user interfaces to enable an administrator of grid computing environment 100 to access each of the grid managers 152, 154, 156, 160, 162,

and 164 individually or all together. The API is static in that it includes only one method, i.e., a string that contains a command typed by the administrator. The API is dynamic because the string can contain many different commands.

In some cases, the grid managers are developed using the Java programming language. In these cases, new commands issued to the grid managers can be supported by loading new or revised Java classes dynamically via classloaders. This dynamic access to code can be done without shutting down grid managers in the grid computing environment 100. Using Java classloaders, each time an instance of a class for a grid manager is generated, the definition and behavior of the class can be updated to provide new functionality to the grid computing environment 100.

Another way to modify the functionality of the grid computing environment 100 dynamically without shutting down the grid computing environment 100 is to change the hierarchical relations between grid managers, remove grid managers, or add new grid managers. The API provided for administration of the grid computing environment 100 is also configured to send strings to individual grid managers with commands to delete existing relations or add new relations.

For administrators of grid computing environment 100, it is useful to visualize the applications and a grid manager on one computer device in the grid computing environment 100 as well as other computer devices running part of the grid management hierarchy in the form of grid managers with one or more levels of inferior relations to the grid manager. The view of these computer devices is referred to as a grid landscape. As shown in FIG. 6, a grid graphical user interface (GUI) 500 for visualization of a grid landscape, such as the grid computing environment 100, includes GUI-elements visualizing an organization of services running on computer devices. The GUI 500 provides a grid-like structure with columns and rows. Rows represent services, which in turn are structured hierarchically with respect to the application where a service belongs to, the type of the service, and the specific service instances. Each service instance row is associated with a place in the grid computing environment 100 representing where it is instantiated. In this context, columns represent the computer devices in the grid landscape. Specifically, GUI 500 has three columns representing three computer devices 12, 14, and 16. GUI 500 shows that grid manager 152 runs on computer device 12 with inferior grid managers 154 and 156 running on computer devices 14 and 16, respectively. GUI

500 also shows internet pricing configurator services running on computer device 12. These internet pricing configurator services include IPC dispatcher 116, IPC server 120, and IPC manager 118.

The GUI 500 is dynamically refreshed with feedback from the grid managers and internet pricing configurator (or other application) services so that new services appear in GUI 500 to an administrator. Similarly, services that are shut down are removed in GUI 500.

As shown in FIG. 7, a grid browser component 600 is a composite graphical user interface (GUI) for browsing grid managers on computer devices in the grid computing environment 100. The component 600 displays a graph with curved edges and vertices. Vertices represent computer devices in the grid computing environment 100 and curved edges represent the directional association of grid managers on two computer devices (vertices) in the grid computing environment 100. This association is hierarchical (i.e., superior/inferior). Each vertex displays the network address of a computer device as well as applications currently running on the computer device. For example, component 600 shows computer devices 12, 14, 16, 18, 20, and 22 with IPC servers 118, 120, 122, and 124. In other examples (not shown), the grid browser component 600 shows non-hierarchical, peer to peer associations of grid managers with non-directional edges representing the associations.

The grid browser component 600 is context sensitive. Depending on the relationship among the grid managers on the computer devices (e.g., superior/inferior), computer devices are traversed in respect to a user's browsing history.

By clicking on a vertex representing a computer device in GUI 600 (e.g., computer device 14), a user can automatically view a grid manager and applications running on the computer device and grid managers having inferior relations to the grid manager using GUI 500. The user can pick a computer device and see relations between its grid manager and other grid managers. This connection between GUIs 500 and 600 is done using software that generates GUIs 500 and 600.

The network 8 can be implemented in a variety of ways. The network 8 includes any kind and any combination of networks such as an Internet, a local area network (LAN) or other local network, a private network, a public network, a plain old telephone system (POTS), or other similar wired or wireless networks. Communications through the network 8 may be secured with a mechanism such as encryption, a security protocol, or other type of similar

mechanism. Communications through the network 8 can include any kind and any combination of communication links such as modem links, Ethernet links, cables, point-to-point links, infrared connections, fiber optic links, wireless links, cellular links, Bluetooth®, satellite links, and other similar links.

5 The network 8 is simplified for ease of explanation. The network 8 can include more or fewer additional elements such as networks, communication links, proxy servers, firewalls or other security mechanisms, Internet Service Providers (ISPs), gatekeepers, gateways, switches, routers, hubs, client terminals, and other elements.

10 Computer devices 12, 14, 16, 18, 20, and 22 communicate over medium 10 using one of many different networking protocols. For instance, one protocol is Transmission Control Protocol/Internet Protocol (TCP/IP) combined with SOAP (Simple Object Access Protocol).

15 Embodiments of the invention can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. Embodiment of the invention can be implemented as a computer program product, i.e., a computer program tangibly embodied in an information carrier, e.g., in a node-readable storage device or in a propagated signal, for execution by, or to control the operation of, data processing apparatus, e.g., a programmable processor, a computer, or multiple computers. A computer program can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, 20 subroutine, or other unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a communication network.

25 Method steps of embodiments of the invention can be performed by one or more programmable processors executing a computer program to perform functions of the invention by operating on input data and generating output. Method steps can also be performed by, and apparatus of the invention can be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit).

30 Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read-only memory or a random access memory or both. The essential elements of a computer are a

processor for executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. Information carriers suitable for embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in special purpose logic circuitry.

To provide for interaction with a user, embodiments of the invention can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input.

Embodiments of the invention can be implemented in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of embodiments of the invention, or any combination of such back-end, middleware, or front-end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

A number of embodiments of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Other embodiments are within the scope of the following claims.

WHAT IS CLAIMED IS:

- 1 1. A method comprising:
 - 2 in a client server network, maintaining systems having grid managers having
 - 3 hierarchical relations, the relations of each grid manager stored in each of the systems.
- 1 2. The method of claim 1 in which each of the relations are classified as superior or
2 inferior.
- 1 3. A system comprising:
 - 2 a network of computer systems, each of the computer systems including a
 - 3 grid management engine, each of the grid managers having hierarchical relations with
 - 4 other grid managers, the relations of each grid manager stored in each of the systems.
- 1 4. The method of claim 3 in which each of the relations are classified as superior or
2 inferior.
- 1 5. A method comprising:
 - 2 in a network, starting an execution of a first service on a first computer, the first
 - 3 service handling at least locating, reserving, allocating, monitoring, and deallocating one
 - 4 or more computational resources for one or more applications using the network;
 - 5 reading, by the first service, a file to inform the first service of a relation with a
 - 6 second service, wherein the first service has a inferior relation with the second service,
 - 7 the inferior relation meaning that the second service can send a query for available
 - 8 computer resources to the first service;
 - 9 establishing a first communication channel from the first service to the second
 - 10 service; and
 - 11 accepting an opening of a second communication channel from the second service
 - 12 to the first service.
- 1 6. The method of claim 5 further comprising:
 - 2 receiving a message to cancel the first service's inferior relation with the second
 - 3 service;
 - 4 closing the first and second communication channels;

5 receiving a message to generate a inferior relation from the first service to a third
 6 service residing in a third computer;
 7 establishing a third communication channel from the second service to the third
 8 service; and
 9 accepting an opening of a fourth communication channel from the third service to
 10 the first service.

1 7. The method of claim 5 wherein establishing a first communication channel further
 2 comprises determining if the second service responds to determining and if not,
 3 establishing a communication channel to the second service after a predetermined time
 4 period.

1 8. A method comprising:

2 in a network, starting an execution of a first service residing in a first computer,
 3 the first service handling at least locating, allocating, monitoring, and deallocating one or
 4 more computational resources for one or more applications using the network;

5 starting an execution of a second service residing in a second computer;

6 reading, by the second service, a file to inform the second service of a relation
 7 with the first service, wherein the second service has a inferior relation with the first
 8 service, wherein the inferior relation indicates that the first service can send a query for
 9 available computer resources to the second service;

10 establishing a first communication channel from the second service to the first
 11 service; and

12 establishing a second communication channel from the first service to the second
 13 service.

1 9. The method of claim 8 further comprising:

2 receiving, by the second service, a message to cancel the second service's relation
 3 with the first service;

4 closing the first communication channel;

5 failing to respond to the second communication channel;

6 receiving a message to create a inferior relation from the second service to a third
7 service;

8 establishing a third communication channel from the second service to the third
9 service; and

10 establishing a fourth communication channel from the second service to the third
11 service.

1 10. A system comprising:

2 two or more computers each configured to run a service, the service handling at
3 least locating, allocating, monitoring, and deallocating one or more computational
4 resources for one or more applications;

5 a network of the services, the network configured such that a first service from the
6 services has a superiorrelation with a second service from the services and the second
7 service has an inferior relation with the first service, wherein the first service is
8 configured to check the status of the second service in the network by waiting for a
9 response to a query from the first service to the second service.

1 11. The system of claim 10 wherein the relation comprises a first communication channel
2 from the first service to the second service and a second communication channel from the
3 second service to the first service.

1 12. The system of claim 10 wherein the first service is further configured to locate the
2 one or more computational resources for the one or more applications by sending a query
3 for available computational resources to the second service.

1 13. The system of claim 10 wherein the second service is further configured to remove its
2 inferior relation with the first service and create a new superior relation with a third service.

ABSTRACT

A method includes, in a grid computing environment, maintaining systems having grid managers having hierarchical relations, the relations of each grid manager stored in each of the systems. Each of these hierarchical relations are classified as superior or inferior.

5

20751446.doc

COMBINED DECLARATION AND POWER OF ATTORNEY

As a below-named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter that is claimed and for which a patent is sought on the invention entitled:

GRID ORGANIZATION

the specification of which:

☒ is attached hereto.

☐ was filed on _____.

☐ under Application No. _____
☒ with Express Mail No. EV331001551US (Application Number not yet known).

☐ was described and claimed in PCT International Application No. _____.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information that is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, Section 1.56(a).

I hereby claim the benefit under Title 35, United States Code, §119(e)(1) of any United States provisional application(s) listed below:

<u>U.S. Serial No.</u>	<u>Filing Date</u>	<u>Status</u>
60/490,818	July 28, 2003	Pending

I hereby claim the benefit under Title 35, United States Code, §120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose all information I know to be material to patentability as defined in Title 37, Code of Federal Regulations, §1.56(a) which became available between the filing date of the prior application and the national or PCT international filing date of this application:

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate or of any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed:

I hereby appoint all registered practitioners associated with Customer Number 32864 to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith, and direct that all correspondence be addressed to:

Customer Number 32864

Direct all telephone calls to KENNETH F. KOZIK, Reg. No. 36,572, at telephone number (817) 542-5070.

☒ For Assigned Inventions: I understand that the purpose of making this appointment is to permit prosecution of patent applications for the above-identified invention for the benefit of my assignee, and that this appointment does not create an attorney-client relationship between me and these appointees.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: EROL BOZAK

Inventor's Signature: 

Date: Nov 10, 2003

Residence Address: Pforzheim

Citizenship: Turkish

Post Office Address: Forststrasse 8

75175 Pforzheim
Germany

Full Name of Inventor: ALEXANDER GEBHART

Inventor's Signature: 

Date: Nov 10, 2003

Residence Address: Bad Schoenborn

Citizenship: Germany

Post Office Address: Rosenweg 5

76689 Bad Schoenborn
Germany

20751907.doc

A Performance Oriented Migration Framework For The Grid *

Sathish S. Vadhiyar and Jack J. Dongarra
 Computer Science Department
 University of Tennessee
 {vss; dongarra}@cs.utk.edu

Abstract

At least three factors in the existing migration frameworks make them less suitable in Grid systems especially when the goal is to improve the response times for individual applications. These factors are the separate policies for suspension and migration of executing applications employed by these migration frameworks, the use of pre-defined conditions for suspension and migration and the lack of knowledge of the remaining execution time of the applications. In this paper we describe a migration framework for performance oriented Grid systems that implements tightly coupled policies for both suspension and migration of executing applications and takes into account both system load and application characteristics. The main goal of our migration framework is to improve the response times for individual applications. We also present some results that demonstrate the usefulness of our migration framework.

1. Introduction

Computational Grids [8] involve large system dynamics such that the ability to migrate executing applications onto different sets of resources assumes great importance. Specifically, the main motivations for migrating applications in Grid systems are to provide fault tolerance and to adapt to load changes on the systems.

In this paper, we focus on migration of applications executing on the distributed and Grid systems when the loads on the system resources change. There are at least two disadvantages in using the existing migration frameworks [11, 16, 19, 9, 11] for improving the response times of executing applications. Due to the separate policies employed by these migration frameworks for suspension of executing applications and migration of the applications to dif-

ferent systems, the applications can incur lengthy waiting times between when they are suspended and when they are restarted on new systems. Secondly, due to the use of pre-defined conditions for suspension and migration and due to the lack of knowledge of the remaining execution time of the applications, the applications can be suspended and migrated even when they are about to finish execution in a short period of time.

In this paper, we describe a framework that defines and implements scheduling policies for migrating applications executing on distributed and Grid systems in response to system load changes. In our framework, the migration of applications depends on

1. the amount of increase or decrease in loads on the resources,
2. the time of the application execution when load is introduced into the system,
3. the performance benefits that can be obtained for the application due to migration.

Our migration framework is primarily intended for rescheduling long running applications that typically execute for several minutes. The migration of applications in our migration framework is dependent on the ability to predict the remaining execution times of the applications which in turn is dependent on the presence of execution models that predict the total execution cost of the applications. The framework has been implemented and tested in the GrADS system [2]. Our test results indicate that our migration framework can help improve the performance of executing applications by more than 30%.

In Section 2, we describe the GrADS system and the life cycle of GrADS applications. In Section 3, we introduce our migration framework by describing the different components for migration. In Section 4, we describe our experiments and provide various results. In Section 5, we present related work in the field of migration. We give concluding remarks and explain our future plans in Section 6.

*This work is supported in part by the National Science Foundation contract GRANT #EIA-9975020, SC #R36505-29200099 and GRANT #EIA-9975015

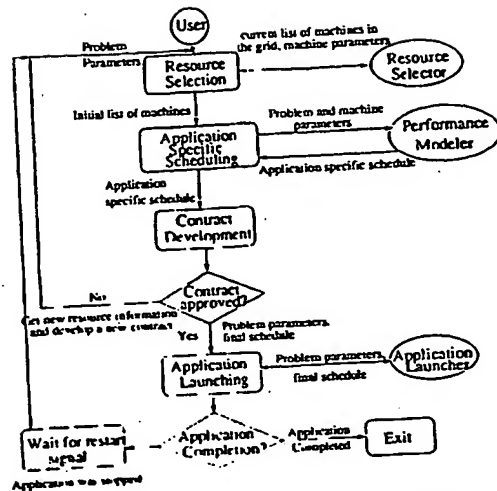


Figure 1. GrADS application manager

2. The GrADS System

GrADS [2] is an ongoing research project involving a number of institutions and its goal is to simplify distributed heterogeneous computing in the same way that the World Wide Web simplified information sharing over the Internet. In the architecture of GrADS, the user wanting to solve a numerical application over the Grid invokes the GrADS application manager. The life cycle of the GrADS application manager is shown in Figure 1. The application manager invokes a com-

As a first step, the application manager invokes a component called Resource Selector. The Resource Selector accesses the Globus Monitoring and Discovery Service (MDS) [7] to retrieve a list of machines in the GrADS testbed that are alive and then contacts the Network Weather Service (NWS) [18] to retrieve system information for the machines. The application manager then invokes a component called Performance Modeler with problem parameters, machines and machine information. The Performance Modeler, using an execution model built specifically for the application, determines the final list of machines for application execution. By employing the application specific execution model, GrADS follows the AppLeS [3] approach to scheduling. The problem parameters and the final list of machines are passed as a contract to a component called Contract Developer. The Contract Developer can either approve or reject the contract. If the contract is rejected, the application manager develops a new contract by starting from the resource selection phase again. If the contract is approved, the application manager passes the problem, its parameters and the final list of machines to Application

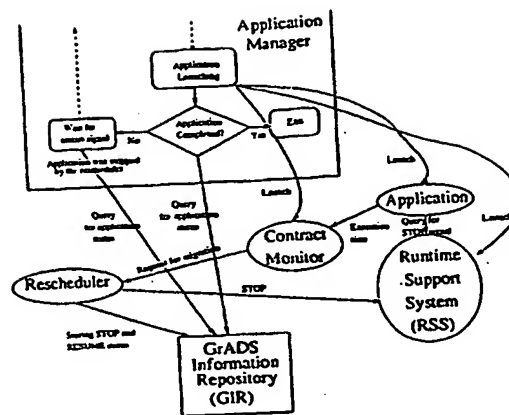


Figure 2. Interactions in Migration framework

Launcher. The Application Launcher spawns the job on the given machines using Globus job management mechanism and also spawns a component called Contract Monitor. The Contract Monitor through an Autopilot mechanism [13] monitors the times taken for different parts of applications. The GrADS architecture also has a GrADS Information Repository (GIR) that maintains the different states of the application manager and the states of the numerical application. After spawning the numerical application through the Application Launcher, the application manager waits for the job to complete. The job can either complete or suspend its execution due to external intervention. These application states are passed to the application manager through the GIR. If the job has completed, the application manager exits, passing success values to the user. If the application is stopped, the application manager waits for a resume signal and then collects new machine information by starting from the resource selection phase again.

3. The Migration Framework

The ability to migrate applications in the GrADS system is implemented by adding a component called *Rescheduler* to the GrADS architecture. The migrating numerical application, *migrator*, the *contract monitor* that monitors the application's progress and the *rescheduler* that decides when to migrate, together form the core of the migration framework. The interactions between the different components involved in the migration framework is illustrated in Figure 2. These components are described in detail in the following subsections.

3.1. The Migrator

We have implemented a user-level checkpointing library called SRS (Stop Restart Software). The application by making calls to SRS possesses the ability to checkpoint data, to be stopped at a particular point in execution, to be restarted and continued later on a different configuration of processors. The SRS library is implemented on top of MPI at the application layer and migration is achieved by clean exit of the entire application and restarting the application over a new configuration of machines. The application interfaces for SRS look similar to CUMULVS [10], but unlike CUMULVS, SRS does not require a PVM virtual machine to be setup on the hosts. Although the method of rescheduling in SRS, by stopping and restarting executing applications, incurs more overhead than process migration techniques [4, 5, 15], the approach followed by SRS allows re-configuration of executing applications and portable across different MPI implementations.

The SRS library consists of 6 main functions: SRS.Init(), SRS.Finish(), SRS.Restart.Value(), SRS.Check_Stop(), SRS.Register() and SRS.Read(). The user calls SRS.Init() and SRS.Finish() in his application after MPI.Init() and before MPI.Finalize() respectively. In order to know if the application is executed in the start or restart mode, the user calls SRS.Restart.Value() that returns 0 and 1 on start and restart modes respectively. The user also calls SRS.Check_Stop() at different phases of the application to check if an external component wants the application to be stopped.

SRS library uses Internet Backplane Protocol (IBP) [12] for storage of the checkpoint data. IBP depots are started on all the machines of the GrADS testbed. The user calls SRS.Register() in his application to register the variables that will be checkpointed by the SRS library. When an external component stops the application, the SRS library checkpoints only those variables that were registered through SRS.Register(). The user reads in the checkpointed data in the restart mode using SRS.Read(). The user, through SRS.Read(), also specifies the previous and current data distributions. By knowing the number of processors and the data distributions used in the previous and current execution of the application, the SRS library automatically performs the appropriate data redistribution. Thus, for example, the user can start his application on 4 processors with block distribution of data, stop the application and restart it on 8 processors with block-cyclic distribution. The details of the SRS API for accomplishing the automatic redistribution of data is beyond the scope of the current discussion.

An external component (e.g., the rescheduler) wanting to stop an executing application interacts with a daemon called Runtime Support System (RSS). RSS exists for the entire

duration of the application and spans across multiple migrations of the application. Before the actual parallel application is started, the RSS is launched by the application launcher on the machine where the user invokes the GrADS application manager. The actual application through the SRS library interacts with RSS to perform some initialization, to check if the application needs to be stopped during SRS.Check_Stop() and to store and retrieve pointers to the checkpointed data.

3.2. Contract Monitor

Contract Monitor is a component that uses the Autopilot infrastructure [13] to monitor the progress of the applications in GrADS. An autopilot manager is started before the launch of the numerical application. The numerical application is instrumented with calls to send the execution times taken for the different phases of the application to the contract monitor. The contract monitor compares the actual execution times with the predicted execution times and calculates the ratio between them. The tolerance limits of the ratio are specified as inputs to the contract monitor.

When a given ratio is greater than the upper tolerance limit, the contract monitor calculates the average of the computed ratios. If the average is greater than the upper tolerance limit, it contacts the rescheduler, requesting for migrating the application. The average of the ratios is used by the contract monitor to contact the rescheduler due to the following reasons:

1. A competing application of short duration on one of the machines may have increased the load on the machine and hence the loss in performance of the application. Contacting the rescheduler for migration on noticing few losses in performance will result in unnecessary migration in this case since the competing application will end soon and the application's performance will be back to normal.
2. The average of the ratios also captures the history of the behavior of the machines on which the application is running.
3. The average of the ratios also takes into account the percentage completed time of application's execution.

If the rescheduler refuses to migrate the application, the contract monitor adjusts its tolerance limits to new values. Similarly when a given ratio is less than the lower tolerance limit, the contract monitor calculates the average of the ratios and adjusts the tolerance limits if the average is less than the lower tolerance limit. The dynamic adjusting of tolerance limits not only reduces the amount of communication between the contract monitor and the rescheduler but also hides the deficiencies in the application-specific execution time model.

3.3. Rescheduler

Rescheduler is the component that evaluates the performance benefits that can be obtained due to the migration of an application and initiates the migration of the application. It operates in two modes: *migration on request* and *opportunistic migration*. When the contract monitor detects intolerable performance loss for an application, it contacts the rescheduler requesting it to migrate the application. This is called migration on request. In other cases, if a GrADS application was recently completed, the rescheduler determines if performance benefits can be obtained for an executing application by migrating it to use the resources that were freed by the completed application. This is called opportunistic rescheduling.

In both cases, the rescheduler first contacts the Network Weather Service (NWS) to get the updated information for the machines in the Grid. It then contacts the application-specific performance modeler to evolve a new schedule for the application. Based on the total percentage completion time for the application and the total predicted execution time for the application with the new schedule, the rescheduler calculates the remaining execution time, *ret_new*, of the application if it were to execute on the machines in the new schedule. The rescheduler also calculates *ret_current*, the remaining execution time of the numerical application if it were to continue executing on the original set of machines. The rescheduler then calculates the rescheduling gain as

$$\text{rescheduling_gain} = \frac{(\text{ret_current} - (\text{ret_new} + 900))}{\text{ret_current}}$$

The number 900 in the numerator of the fraction is the worst case time in seconds needed to reschedule the application. The various times involved in rescheduling is given in Table 1. The times shown in Table 1 were obtained by conducting a number of experiments with different problem sizes and obtaining the maximum times for each phases of rescheduling. Thus the rescheduling strategy adopts pessimistic approach for rescheduling where migration of applications will be avoided in certain cases where migration can yield performance benefits.

If the rescheduling gain is greater than 30%, the rescheduler sends STOP signal to the application, and stores the stop status in GIR. The application manager then waits for the RESUME signal. The rescheduler stores the RESUME value in the GIR thus prompting the application manager to evolve a new schedule and restart the application on the new schedule. If the rescheduling gain is less than 30% and if the rescheduler is operating in the *migration on request* mode, the rescheduler contacts the contract monitor prompting the contract monitor to adjust its tolerance limits.

The rescheduling threshold [17] which the performance gain due to rescheduling must cross for rescheduling to

Rescheduling Phase	Time (secs.)
Writing checkpoints	40
Waiting for NWS to update information	90
Time for application manager to get new resource information from NWS	120
Evolving new application-level schedule	80
Other grid overhead	10
Starting application	60
Reading checkpoints and Data redistribution	500
Total	900

Table 1. Times for rescheduling phases

yield significant performance benefits depends on the load dynamics of the system resources, the accuracy of the measurements of resource information and may also depend on the particular application for which rescheduling is made. Since the measurements made by NWS are fairly accurate, the rescheduling threshold for our experiments depended only on the load dynamics of the system resources. By means of trial-and-error experiments we determined the rescheduling threshold for our testbed to be 30%.

4. Experiments and Results

The GrADS experimental testbed consists of about 40 machines that reside in institutions across United States including University of Tennessee, University of Illinois, University of California at San Diego, Rice University etc. For the sake of clarity, our experimental testbed consists of two clusters, one in University of Tennessee and another in University of Illinois, Urbana-Champaign. The Tennessee cluster consists of 8 933 MHz dual-processor Pentium III machines running Linux and connected to each other by 100 Mb switched Ethernet. The Illinois cluster consists of 16 450 MHz single-processor Pentium II machines running Linux and connected to each other by 1.28 Gbit/second full duplex myrinet. The two clusters are connected by means of Internet.

About 5 applications, namely, ScaLAPACK LU and QR factorizations, ScaLAPACK eigen value problems, PETSC, CG application and heat equation solver have been integrated into the migration framework by instrumenting the applications with SRS calls and writing performance models for the applications. In general, our migration framework is suitable for iterative parallel applications for which performance models predicting the execution costs can be written. In our experiments, ScaLAPACK QR factorization was used as the end application. The data that were checkpointed by the SRS library for the application included the matrix, A and the right-hand side vector, B.

4.1. Migration on Request

In all the experiments in this section, 4 Tennessee machines and 8 Illinois machines were used. A given matrix size for the QR factorization problem was input to the application manager. For large problem sizes, the computation time dominates the communication time for the ScaLAPACK application. Since the Tennessee machines have higher computing power than the Illinois machines, the application manager by means of the performance modeler chose the 4 Tennessee machines for the end application run. A few minutes after the start of the end application, artificial load is introduced into the 4 Tennessee machines. This artificial load is achieved by executing a certain number of loading programs on each of the Tennessee machines. The loading program used was a sequential C code that consists of a single looping statement that loops forever. This program was compiled without any optimization in order to achieve the loading effect.

Due to the loss in predicted performance caused by the artificial load, the contract monitor requested the rescheduler to migrate the application. The rescheduler evaluated the potential performance benefits that can be obtained by migrating the application to the 8 Illinois machines and either migrated the application or allowed the application to continue on the 4 Tennessee machines. The rescheduler was operated in two modes - a default and a non-default mode. The normal operation of the rescheduler is its default mode and the non-default mode of the rescheduler is when the rescheduler code was modified to force the application to either migrate or continue on the same set of resources. Thus in cases when the default mode of the rescheduler was to migrate the application, the non-default mode was to continue the application on the same set of resources and in cases when the default mode of the rescheduler was to not migrate the application, the non-default mode was to force the rescheduler to migrate the application by adjusting the rescheduling cost parameters. For each experimental run, results were obtained for both when rescheduler was operated in the default and non-default mode. This allowed us to compare both scenarios and to verify if the rescheduler made the right decision.

Three parameters were involved in each set of experiments - the size of the matrices, the amount of load and the time after the start of the application when the load was introduced into the system. The following three sets of experiments were obtained by fixing two of the parameters and varying the other parameter.

In the first set of experiments, the artificial load consisting of 10 loading programs was introduced into the system 5 minutes after the start of the end application. The bar chart in Figure 3 was obtained by varying the size of the matrices, i.e. the problem size on the x-axis. The y-axis

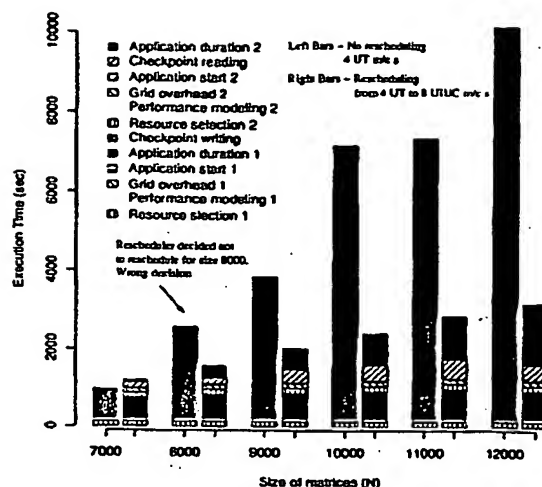


Figure 3. Problem Sizes and Migration

represents the execution time in seconds of the entire problem including the Grid overhead. For each problem size, the bar on the left represents the execution time when the application was not migrated and the bar on the right represents the execution time when the application was migrated.

Several points can be observed from Figure 3. The time for reading checkpoints occupied most of the rescheduling cost since it involves moving data across the Internet from Tennessee to Illinois and redistribution of data from 4 to 8 processors. On the other hand, the time for writing checkpoints is insignificant since the checkpoints are written to local disks. The rescheduling benefits are more for large problem sizes since the remaining lifetime of the end application when load is introduced is larger for larger problem sizes. There is a particular size of the problem below which the migrating cost overshadows the performance benefit due to rescheduling. Except for matrix size 8000, the rescheduler made the correct decision for all matrix sizes. For matrix size 8000, the rescheduler assumed a worst-case rescheduling cost of 900 seconds while the actual rescheduling cost was close to about 420 seconds. Thus the rescheduler evaluated the performance benefit to be negligible while the actual scenario points to the contrary. Thus the pessimistic approach followed by using a worst-case rescheduling cost in the rescheduler will lead to underestimating the performance benefits due to rescheduling in some cases.

In the second set of experiments, matrix size 12000 was chosen for the end application and artificial load was in-

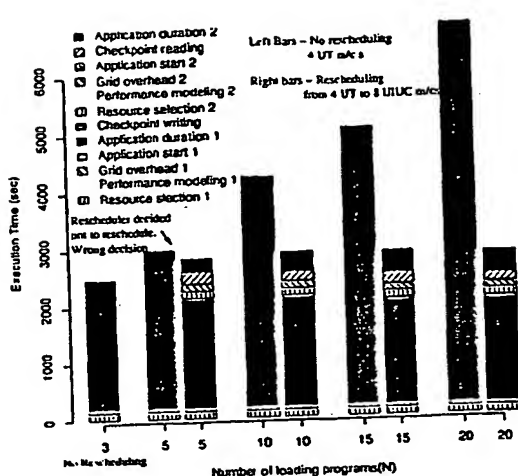


Figure 4. Load Amount and Migration

Introduced 20 minutes into the execution of the application. In this set of experiments, the amount of artificial load was varied by varying the number of loading programs that were executed. In Figure 4, the x-axis represents the number of loading programs and the y-axis represents the execution time in seconds. For each amount of load, the bar on the left represents the case when the application was continued on 4 Tennessee machines and the bar on the right represents the case when the application was migrated to 8 Illinois machines.

In the third set of experiments, shown in Figure 5, equal amount of load consisting of 7 loading programs was introduced at different points of execution of the end application for the same problem of matrix size 12000. The x-axis represents the elapsed time in minutes of the execution of end application when the load was introduced. The y-axis represents the total execution time in seconds. Similar to the previous experiments, the bars on the left denote the cases when the application was not rescheduled and the bars on the right represent the cases when the application was rescheduled. From Figures 4 and 5, we observe that the performance benefits due to rescheduling increase with the amount of load and decrease as the load is introduced later into the program execution.

4.2. Opportunistic Migration

In this set of experiments, we illustrate opportunistic migration in which the rescheduler tries to migrate an execut-

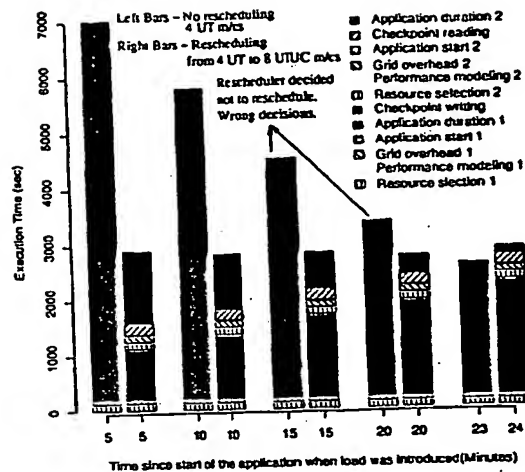


Figure 5. Load Introduction Time and Migration

ing application when some other application completes. For these experiments, two problems were involved. For the first problem, matrix size of 14000 was input to the application manager and 6 Tennessee machines were made available. The application manager, through the performance modeler chose the 6 machines for the end application run. Two minutes after the start of the end application for the first problem, a second problem of a given matrix size was input to the application manager. For the second problem, the 6 Tennessee machines on which the first problem was executing and 2 Illinois machines were made available. Due to the presence of the first problem, the 6 Tennessee machines alone were insufficient to accommodate the second problem. Hence the performance modeler chose the 6 Tennessee machines and 2 Illinois machines for the end application and the actual application run involved communication across the Internet.

In the middle of the execution of the second application, the first application completed and hence the second application can be potentially migrated to use only the 6 Tennessee machines. Although this involved constricting the number of processors of the second application from 8 to 6, there can be potential performance benefits due to the non-involvement of Internet. The rescheduler evaluated the potential performance benefits due to migration and made an appropriate decision.

Figure 6 shows the results for two illustrative cases when

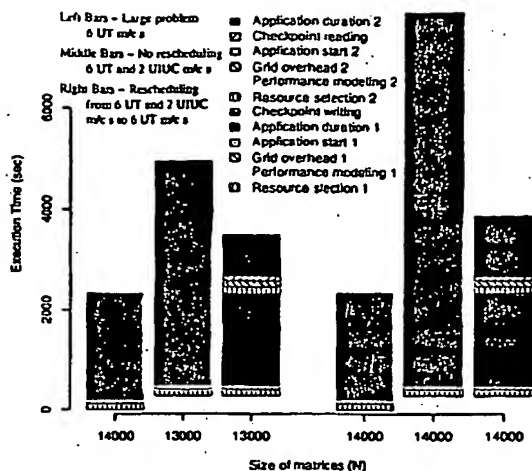


Figure 6. Opportunistic Migration

matrix sizes of the second application were 13000 and 14000. The x-axis represents the matrix sizes and the y-axis represents the execution time in seconds. For each application run, three bars are shown. The bar on the left represents the execution time for the first application that was executed on 6 Tennessee machines. The middle bar represents the execution time of the second application when the entire application was executed on 6 Tennessee and 2 Illinois machines. The bar on the right represents the execution time of the second application, when the application was initially executed on 6 Tennessee and 2 Illinois machines and later migrated to execute on only 6 Tennessee machines when the first application completed.

In both problem cases, matrix sizes 13000 and 14000, for the second problem, the rescheduler made the correct decision of migrating the application. We also find that for both problem cases, the second application was almost immediately rescheduled after the completion of the first application.

5. Related Work

Different systems have been implemented to migrate executing applications onto different sets of resources. These systems migrate applications either to efficiently use under-utilized resources [14, 5, 4, 16, 6], to provide fault resilience [1] or to reduce the obtrusiveness to workstation owner [1, 11]. The particular projects that are closely related to our

work are Dynamite [16], MARS [9], LSF [19] and Condor [11].

The Dynamite system [16] based on Dynamic PVM [6] migrates applications when the loads of certain machines are under-utilized or over-utilized as defined by application-specified thresholds. Although this method takes into account application-specific characteristics it does not necessarily evaluate the remaining execution time of the application and the resulting performance benefits due to migration. MARS [9] migrates applications taking into account both the system loads and application characteristics. But the migration decisions are made only at different phases of the applications unlike our migration framework where the applications are continuously monitored and migration decisions are made whenever the applications are not making sufficient progress.

In LSF [19], jobs can be submitted to queues which have pre-defined migration thresholds. A job can be suspended when the load of the resource increases beyond a particular limit and can be migrated when the time since the suspension becomes higher than the migration threshold for the queue. Thus LSF suspends jobs to maintain the load level of the resources while our migration framework suspends jobs only when it is able to find better resources where the jobs can be migrated. By adopting a strict approach to suspending jobs based on pre-defined system limits, LSF gives less priority to the stage of the application execution, whereas our migration framework suspends an application only when the application has large enough remaining execution time so that performance benefits can be obtained due to migration. And lastly, due to the separation of the suspension and migration decisions, a suspended application in LSF can wait for a long time before it restarts executing on a suitable resource. In our migration framework, a suspended application is immediately restarted due to the tight coupling of suspension and migration decisions.

Of the Grid computing systems, only Condor [11] seems to migrate applications under workload changes. Condor provides powerful and flexible ClassAd mechanism by means of which the administrator of resources can define policies for allowing jobs to execute on the resources, suspending the jobs and vacating the jobs from the resources. The fundamental philosophy of Condor is to increase the throughput of long running jobs and also respect the ownership of the resource administrators. The main goal of our migration framework is to increase the response times of individual applications. Similar to LSF, Condor also separates the suspension and migration decisions and hence has the same problems mentioned for LSF in taking into account the performance benefits of migrating the applications. Unlike our metascheduler framework, the Condor system does not possess the knowledge about the remaining execution time of the applications. Thus suspension and migrating

decisions can be invoked frequently in Condor based on system load changes. This may be less desirable in Grid systems where system load dynamics are fairly high.

6. Conclusions and Future Work

Many existing migration frameworks that migrate applications under loading conditions implement simple policies that cannot be applied to Grid systems. We have implemented a migration framework that takes into account both the system load and application characteristics. Experiments were conducted and results were presented to demonstrate the capabilities of the migration framework.

Of the various costs involved in rescheduling, the cost for data redistribution is the only significant cost that depends on the number and amount of checkpointed data, the data distributions used for the data and the current and future processors sets for the application. We are planning to modify the SRS library and the interactions in the migration framework so that the redistribution cost can be dynamically calculated. Also, instead of fixing the rescheduler threshold at 30%, our future work will involve determining the rescheduling threshold dynamically based on the dynamic observation of load behavior on the system resources. Finally, we propose to investigate the usefulness of our approach for complex applications involving multiple components and/or written in multi-programming languages.

References

- [1] J. Arabe, A. Lowekamp, E. Seligman, M. Starkey, and P. Stephan. Dome: Parallel Programming in a Heterogeneous Multi-User Environment. *Supercomputing*, 1995.
- [2] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Applications and Supercomputing*, 15(4):327-344, Winter 2001.
- [3] F. Berman and R. Wolski. The AppLeS Project: A Status Report. *Proceedings of the 8th NEC Research Symposium*, May 1997.
- [4] J. Casas, D. Clark, P. Galbiati, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MIST: PVM with Transparent Migration and Checkpointing, 1995.
- [5] J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MPVM: A Migration Transparent Version of PVM. Technical Report CSE-95-002, 1, 1995.
- [6] L. Dikken, F. van der Linden, J. J. J. Vesseur, and P. M. A. Sloot. DynamicPVM: Dynamic Load Balancing on Parallel Systems. In W. Gentzsch and U. Harms, editors, *Lecture notes in computer science 797, High Performance Computing and Networking*, volume Proceedings Volume II, Networking and Tools, pages 273-277, Munich, Germany, April 1994. Springer Verlag.
- [7] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. volume Proc. 6th IEEE Symp. on High-Performance Distributed Computing, pages 365-375, 1997.
- [8] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, ISBN 1-55860-475-8, 1999.
- [9] J. Gehring and A. Reinefeld. MARS - A Framework for Minimizing the Job Execution Time in a Metacomputing Environment. *Future Generation Computer Systems*, 12(1):87-99, 1996.
- [10] G. A. Geist, J. A. Kohl, and P. M. Papadopoulos. CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications. *International Journal of High Performance Computing Applications*, 11(3):224-236, August 1997.
- [11] M. Litzkow, M. Livny, and M. Mutka. Condor - a Hunter for Idle Workstations. *Proc. 8th Intl. Conf. on Distributed Computing Systems*, pages 104-111, 1988.
- [12] J. S. Plank, M. Beck, W. R. Elwasif, T. Moore, M. Swamy, and R. Wolski. The Internet Backplane Protocol: Storage in the Network. *NetStore99: The Network Storage Symposium*, 1999.
- [13] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: Adaptive Control of Distributed Applications. *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing*, July 1998.
- [14] K. Saqabi, S. Otto, and J. Walpole. Gang Scheduling in Heterogeneous Distributed Systems. Technical report, OGI, 1994.
- [15] G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)*, pages 526-531, Honolulu, Hawaii, 1996.
- [16] G. van Albada, J. Clinckemaelle, A. Emmen, J. Gehring, O. Heinz, F. van der Linden, B. Overeinder, A. Reinefeld, and P. Sloot. Dynamite - Blasting Obstacles to Parallel Cluster Computing. In P.M.A. Sloot and M. Bubak and A.G. Hoekstra and L.O. Hertzberger, editors, *High-Performance Computing and Networking (HPCN Europe '99)*, Amsterdam, The Netherlands, in series *Lecture Notes in Computer Science*, nr 1593, Springer-Verlag, Berlin, ISBN 3-540-65821-1., pages 300-310, April 1995.
- [17] R. Wolski, G. Shao, and F. Berman. Predicting the Cost of Redistribution in Scheduling. *Proceedings of 8th SIAM Conference on Parallel Processing for Scientific Computing*, March 1997.
- [18] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757-768, October 1999.
- [19] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software - Practice and Experience*, 23(12):1305-1336, December 1993.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☒ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☒ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

THIS PAGE BLANK (USPTO)

#42
4

XP-002306969

CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids

Gregor von Laszewski
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439, U.S.A.
gregor@mcs.anl.gov

Ian Foster
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439, U.S.A.
foster@mcs.anl.gov

Jarek Gawor
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439, U.S.A.
gawor@mcs.anl.gov

ABSTRACT

Emerging national-scale "Computational Grid" infrastructures are deploying advanced services beyond those taken for granted in today's Internet: for example, authentication, remote access to computers, resource management, and directory services. The availability of these services represents both an opportunity and a challenge for the application developer: an opportunity because they enable access to remote resources in new ways, a challenge because these services may not be compatible with the commodity distributed-computing technologies used for application development. The Commodity Grid project is working to overcome this difficulty by creating what we call Commodity Grid Toolkits (CoG Kits) that define mappings and interfaces between Grid and particular commodity frameworks. In this paper, we explain why CoG Kits are important, describe the design and implementation of a Java CoG Kit, and use examples to illustrate how CoG Kits can enable new approaches to application development based on the integrated use of commodity and Grid technologies.

Categories and Subject Descriptors

D.1.3 [Software Engineering]: Concurrent Programming—*Distributed programming*; D.2.6 [Software Engineering]: Programming Environments—*Graphical environments*; D.2.6 [Software Engineering]: Programming Environments—*Programmer workbench*

1. INTRODUCTION

The explosive growth of the Internet and of distributed computing in general has led to rapid technology development in several domains. In the world of commodity computing, a broad spectrum of distributed computing technologies (i.e., Web protocols [16], Java [14], JINI [1], CORBA [4], DCOM [20], etc.) has emerged with revolutionary effects on how we access and process information. Simultaneously, the high-

performance computing community has taken big steps toward the creation of so-called *Grids* [7], advanced infrastructures designed to enable the coordinated use of distributed high-end resources for scientific problem solving.

These two worlds of what we will call "commodity" and "Grid" computing have evolved in parallel, with different goals leading to different emphases and technology solutions. For example, commodity technologies tend to focus on issues of scalability, component composition, and desktop presentation, while Grid developers emphasize end-to-end performance, advanced network services, and support for unique resources such as supercomputers. The results of this parallel evolution are multiple technology sets with some overlaps, much complementarity, and some obvious gaps.

In this context, we believe that it is timely to investigate how the worlds of commodity and Grid computing can be combined. Hence, we have established the *Commodity Grid (CoG) project*, with the twin goals of (a) enabling developers of Grid applications to exploit commodity technologies wherever possible and (b) exporting Grid technologies to commodity computing (or, equivalently, identifying modifications or extensions to commodity technologies that can render them more useful for Grid applications).

A first activity being undertaken within the CoG project is the design and development of a set of Commodity Grid Toolkits (CoG Kits), which we define as follows:

Definition: A Commodity Grid Toolkit (CoG Kit) defines and implements a set of general components that map Grid functionality into a commodity environment/framework.

Hence, we can imagine a Web/CGI CoG Kit, a Java CoG Kit, a CORBA CoG Kit, a DCOM CoG Kit, and so on. In each case, the benefit of the CoG Kit is that it enables application developers to exploit advanced Grid services (resource management, security, resource discovery) while developing higher-level components in terms of the familiar and powerful application development frameworks provided by commodity technologies. In each case, we also face the challenge of developing appropriate interfaces between Grid and commodity concepts and technologies—and, if similar

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Java 2000 San Francisco CA USA
Copyright ACM 2000 1-58113-288-3/00/6...\$5.00

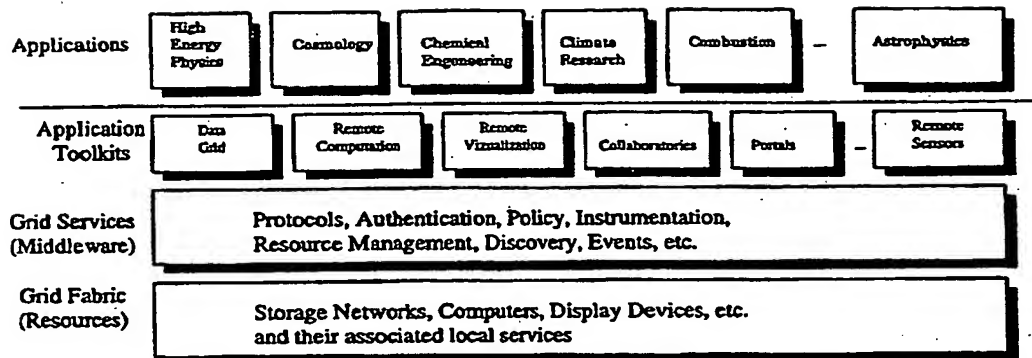


Figure 1: The integrated Grid architecture has four main categories.

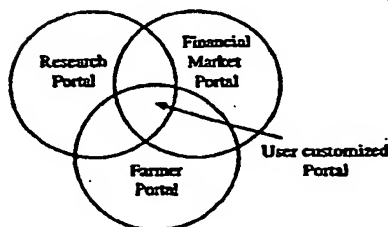


Figure 2: Multiple portals provide access to overlapping functionality, with a particular portal specialized to the requirements of its user.

ble by scientists, allowing for feedback to check for model accuracy. Access points to the portal include computer terminals in electronically enhanced farm buildings and also specialized input and output devices that allow for the installation in, for example, a lightweight wireless device to access a useful subset of the information in the field. The farmer's portal also provides access to other services and information sources, for example, financial market monitoring services that observe the fluctuation of the value of the crops and give advice that may result in greater profits (Figure 2).

3.2 Science Portal Requirements

The creation of science portals such as those just described requires the integration of many technologies from different fields. We will typically provide access to a wide variety of data; hence, we must be able to *access and communicate with a wide range of information sources*. The complex calculations performed on this data requires the ability to access compute resources with significant computational resources. We may also require access to proprietary software loaded on remote machines. Thus, the *ability to incorporate remote computational resources* is required. Interactive use

can require that computational and data resources be accessed via high-performance networks; we would also like to be able to enforce *performance guarantees* for data transfers and computations.

The success of a science portal is also measured by its usability and acceptance in the community. Hence, we require environments that allow rapid prototyping of both complete applications and new components that can be shared with other users. The *ability to rapidly create portable user interfaces* is particularly critical. These requirements overlap strongly with two types of technology:

- *Commodity technologies* that emphasize ease of use and code reuse in local (especially desktop) environments: GUI components, component libraries, scripting languages, industry-accepted distributed computing frameworks, industrial-strength database servers, object-oriented programming languages and frameworks, and the like.
- *Grid technologies* that emphasize effective operation in large-scale, multi-institutional, wide area environments: access to remote computation, information services, high-speed data transfers, special protocols (e.g., multicast), and gateways to local authentication schemes.

These considerations lead to the question that has motivated the research reported in this paper: How can commodity and Grid technologies interface and integrate so as to adhere interoperability — and, ideally, to enhance the capabilities of both? For example, we might decide to use CORBA for application development, but also want to use Grid services for scheduling and managing computations on a supercomputer. Or, if we are using Java, then Jini might appear to be a good mechanism for resource discovery: but then we face the problem of accessing data stored in the extensive (currently LDAP-based) Grid information service. The interactions can be complex and require significant effort by thought to get right. Yet the technology base that exists in each case is sufficiently large and robust that exploiting these

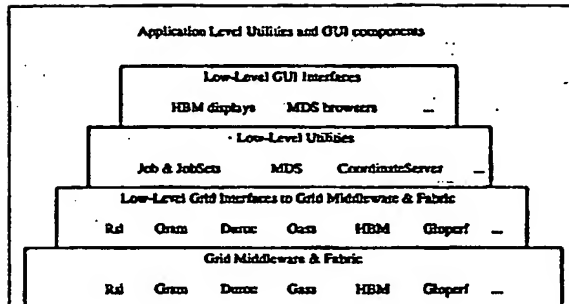


Figure 4: Applications and more complex components can be built with the help of the CoG Kit. Components are classified here based on their role.

6. JAVA COG KIT IMPLEMENTATION

Figure 5 shows how our Java CoG Kit is used in practice. This Java program skeleton forms part of a Climate Portal; it demonstrates how simple it is to build portal-specific services when accessing a variety of basic Grid services through the Java CoG Kit. In this example, an appropriate machine is selected for execution, data for an instantiation of the climate model is located and downloaded to the machine, and the climate model is executed on that machine. The program generates an output file in GrADS [12] format, a well-known format for storing three-dimensional climate related data. Throughout the remainder of paper we will expand this example as we introduce various Java CoG Kit components.

6.1 Low-Level Grid Mappings

In this section we enumerate a subset of packages that provide the interface to the low-level Grid services and application interfaces. These packages are used by many users to develop Java-based programs in the Grid. We will describe only the general functionality of these packages, as it is beyond the scope of this paper to explain every class and method. For a complete list of the classes and methods we refer to the distribution [27].

RSL. The package *org.globus.rsl* provides methods for creating, manipulating, and checking the validity of the RSL expressions used in Globus [11] to express resource requirements. As shown in Step 3 of Figure 5, the arguments to a new call to include parameters that specify both characteristics of the required resources and properties of the computation.

GRAM. The package *org.globus.gram* provides a mapping to the Globus GRAM services [10], which allow users to schedule and manage remote computations. The classes and methods distributed allow users to submit jobs, bind to already submitted jobs, and cancel jobs on remote computers. Other methods allow users to determine whether they can submit jobs to a specific resource (through a Globus gatekeeper) and to monitor the job status (*pending*, *active*, *failed*, *done*, and *suspended*).

```
// Step 0. Initialization
MDS mds=new MDS("www.globus.org",
389,"v=Grid");

// Step 1. Search for an available machine
result = mds.search
("(objectclass=GridComputeResource)
(freenodes=64))","contact");

//Step 1.a) Select a machine
machineContact=<select the machine with
minimal execution time from
the contacts that are
returned in result>

// Step 2. Prepare the data for the experiment

// Step 2.a) Search for climate data and return
// attributes: server,port,directory,file
dn = mds.search
("(objectclass=ClimateData)(year=1999)
(region=midwest","dn", MDS.SubtreeScope);
result = mds.lookup(dn,"server port directory
file");

// Step 2.b) download the data to the machine
url = result.get("server")+ "/"
+ result.get("port")+ "/"
+ result.get("directory")+ "/"
+ result.get("file");
data = server.fetch(url, machineContact);

// Step 3. Prepare a description for running the
model
RSL rsl = new RSL("(executable=climateModel)
(processors=64)
(arguments=grads)(arguments=-out map.grads)
(arguments=-in " + data.filename + ")");

// Step 4. Submit the program
GramJob job = new GramJob();
job.addListener(new GramJobListener() {
public void stateChanged(GramJob job) {
// react to job state changes
}
});
try{
job.request(machineContact, rsl);
} catch (GramException e) {
problem submitting the job
}
```

Figure 5: This sample script demonstrates how we access basic Grid services with the help of the Java CoG Kit. Here data for a climate model is located, an appropriate machine is selected, and the climate model is executed on that machine.

```
// Step 0: Initialize the table
MDSsearchTable table =
    new MDSsearchTable(mds);

// Step 1: perform a search in the MDS
// to request data to be displayed
table.search("(objectclass=GridComputeResources)",
    "hn gramversion contact");

// Step 2: display and update the table
table.show();

// Step 3: return the selection
String machineContact=
    table.getSelection("contact");
```

Figure 8: The program shows the ease of use of the Graphical User Interface for selecting a Grid contact string. (compare Figure 7).

sults of MDS search queries (Figures 7 and 8), trees that display the directory information tree of the MDS, and tables to display HBM and network performance data. Each component can be customized and is available as JavaBean. In future releases of the Java CoG Kit it will be possible to integrate the bean in a Java-based GUI composition tool such as JBuilder or VisualCafe.

6.4 High-Level Graphical Application

High-level graphical applications combine a variety of CoG Kit components to deliver a single application or applet. Naturally, these applications can be combined in order to provide even greater functionality. The user should select the tools that seem appropriate for the task. To demonstrate the range of applications, we have included a set of screen dumps that highlight the look and feel of some applications developed to date.

GECCO. The Graph Enabled Console COmponent (GECCO) is a graphical tool for specifying and monitoring the execution of sets of tasks with dependencies between them [26][24]. Specifically it allows one to

1. specify the jobs and their dependencies graphically or with the help of an XML-based configuration file;
2. debug the specification in order to find erroneous specification strings before the job is submitted; and
3. execute and monitor the job graphically and with the help of a log file.

As shown in Figure 9 each job is represented as a node in the graph. A job is executed as soon as its predecessors are reported as having successfully completed. The state of a job is animated with colors. It is possible to modify the specification of the job while clicking on the node: A specification window pops up allowing the user to edit the RSL, the label, and other parameters. Editing can also be performed during runtime (job execution), hence providing for simple computational steering.

GRC. A second example of a high-level application component is an interactive Graphical Resource Co-allocator

(GRC) illustrated in Figure 10 [5]. This Java application allows the user to build a network representing the resources required for an application and to describe how the resources should be used. A combination of automatic and manual techniques is then used to guide resource selection, eventually generating an RSL specification for the resources in question. MDS services are used to automatically find candidate sets of resources that meet the user's constraints. The user then manually selects one of the resource sets or requests a further search for candidates. Once the user finds a suitable set of resources, the GRAM or DUROC client libraries are used to execute, monitor, and possibly terminate the application(s) (compare Figure 10).

7. FUTURE APPLICATIONS

The availability of the Java CoG Kit has several advantages for developing future Grid-based applications. The assumed platform independence of Java and its increased popularity provide the basis of a promising platform in the near future. Furthermore, since Java is well established on the Windows operating system, it seems an obvious candidate for delivering a Globus server-side implementation, hence allowing jobs to be submitted to any NT machine as long as it is integrated in the Grid. More straightforward is the development of a Globus thin-client, which constitutes only of the necessary security routines and the communication routines to communicate with a Globus server. All previous releases of CoG components used a pull model to inquire about the state of a submitted job. Since we have changed the model to use listeners, it is now easier to write threaded Grid-based Java applications based on a push model. Projects that will benefit from this approach are, for example, Gateway [9] and Webflow [28].

The latest Globus system to relies in many cases on the HTTP protocol, hence it is possible to integrate such a thin-client as part of a Web browser to allow submission through web pages. Projects like WebSubmit [19] and Hotpage [23] will profit from this change. Making some components available as Java Beans and integrating them into common off-the-shelf Java GUI building tools will provide a Grid development environment that allows Grid programming with ease. As a result of the availability of the Java CoG Kit, recent efforts to standardize the Globus delegation model in cooperation with the development of the Java CoG Kit will allow a much easier integration in commodity technology in future.

8. SUMMARY

Commodity distributed-computing technologies enable the rapid construction of sophisticated client-server applications. Grid technologies provide advanced network services for large-scale, wide area, multi-institutional environments and for applications that require the coordinated use of multiple resources. In the Commodity Grid project, we seek to bridge these two worlds so as to enable advanced applications that can benefit from both Grid services and sophisticated commodity development environments.

The Java Commodity Grid Toolkit (CoG Kit) described in this paper represents a first attempt at creating of such a bridge. Building on experience gained over the past three years with the use of Java in Grid environments, we have

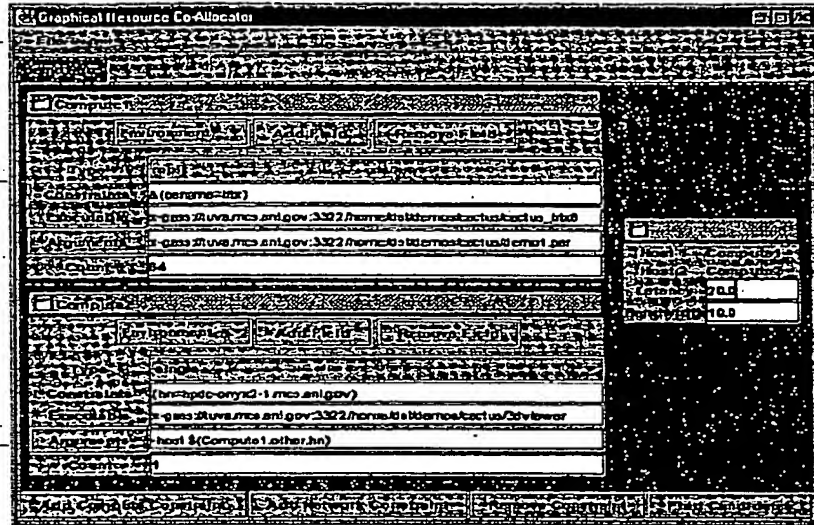


Figure 10: The GRC allows to select a compute resource for scheduling a job interactively from a set of automatically derived machines that fulfill a user-specific constraint.

defined a rich set of classes that provide the Java programmer with access to basic Grid services, enhanced services suitable for the definition of desktop problem solving environments, and a range of GUI elements. Initial experiences with these components have been positive. It has proved possible to recast major Grid services in Java terms without compromising on functionality. Some substantial Java CoG Kit applications have been developed, and reactions from users have been positive.

Our future work will involve the integration of more advanced services into the Java CoG Kit and the creation of other CoG Kits, with CORBA, DCOM, and Python being early priorities. We also hope to gain a better understanding of where changes to commodity or Grid technologies can facilitate interoperability and of where commodity technologies can be exploited in Grid environments.

9. AVAILABILITY

The Java Cog Kit is available in alpha release form the CoG Kit Web pages [27]. The release of the components is done gradually to assure the necessary quality control of the delivered packages, classes, and methods. At present, the main distribution contains the low-level components. Besides the components described in this paper, we have an implementation of network based quality-of-service methods. We expect that this package will be released as soon as the Globus toolkit API for this area is frozen. For more release notes, we refer to the Web page <http://www.globus.org/cog>.

10. ACKNOWLEDGMENTS

Many technologies and research projects are related to and important for the development of the CoG Kits. Some of

them can be found in [3]. We are grateful to members of the NCSA Alliance for enlightening discussions on these topics; in particular, we thank Jay Alameda, Dennis Gannon, Geoffrey C. Fox [8], and Mary Pietrowicz. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the National Science Foundation; and by the NASA Information Power Grid program.

11. ADDITIONAL AUTHORS

Warren Smith, and Steve Tuecke

12. REFERENCES

- [1] K. Arnold, B. Osullivan, R. W. Scheiffer, J. Waldo, A. Wollrath, and B. O'Sullivan. *The Jini Specification*. The Java Technology Series. Addison-Wesley, June 1999.
- [2] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A data movement and access service for wide area computing systems. In *Proc. IOPADS'99*. ACM Press, 1999.
- [3] Computing Portals: Project Catalog. <http://www.computingportals.org/projects>, 1999.
- [4] CORBA 2.0/IIOP Specification. <http://www.omg.org/corba/c2indx.htm>.
- [5] K. Czajkowski, I. Foster, and C. Kesselman. Co-allocation services for computational grids. In *Proc. 8th IEEE Symp. on High Performance*

XP-002308032

Designing Grid-based Problem Solving Environments and Portals

Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane, Nell Rehn, Mike Russell

Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, U.S.A.

gregor@mcs.anl.gov

Abstract

Building Problem Solving environments in the emerging national-scale Computational Grid infrastructure is a challenging task. Accessing advanced Grid services, such as authentication, remote access to computers, resource management, and directory services, is usually not a simple matter for problem solving environment developers. The Commodity Grid project is working to overcome this difficulty by creating what we call Commodity Grid Toolkits (CoG Kits) that define mappings and interfaces between the Grid and particular commodity frameworks familiar to problem solving environment developers. In this paper, we explain why CoG Kits are important for problem solving environment developers, describe the design and implementation of a Java CoG Kit, and use examples to illustrate how CoG Kits can enable new approaches to application development based on the integrated use of commodity and Grid technologies.

1. Introduction

The development of next-generation problem solving environments (PSEs)[12] is influenced by rapid advances in the world of commodity computing and the emerging national-scale Computational Grid. The explosive growth of the Internet and of distributed computing in general has led to significant technology improvements in several domains that are important for the development of PSEs accessing large-scale computational resources. In the world of commodity computing, a broad spectrum of distributed computing technologies (Web protocols, Java, JINI, CORBA, DCOM, etc.) has emerged, with revolutionary effects on how we access and process information. Simultaneously, the high-performance computing community has taken big steps toward the creation of so-called Grids, advanced infrastructures designed to enable the coordinated use of distributed high-end resources for scientific problem solving.

These two worlds of what we will call "commodity" and "Grid" computing have evolved in parallel, with different goals leading to different emphases and technology solutions. For example, commodity technologies tend to focus on issues of scalability, component composition, and desktop presentation, while Grid developers emphasize end-to-end performance, advanced network services, and support for unique resources such as supercomputers. The results of this parallel evolution are multiple technology sets with some overlaps, much complementarity, and some obvious gaps.

In this context, we have established the *Commodity Grid (CoG) project*, with the twin goals of (a) enabling developers of PSEs to exploit commodity technologies wherever possible and (b) exporting Grid technologies to commodity computing for easy integration in PSEs.

A first activity being undertaken within the CoG project is the design and development of a set of Commodity Grid Toolkits (CoG Kits), that define and implement a set of general components that map Grid functionality into a commodity environment/framework. Hence, we can imagine a Web/CGI CoG Kit, a Java CoG Kit, a CORBA CoG Kit, a DCOM CoG Kit, and so on. In each case, the benefit of the CoG Kit is that it enables application developers to exploit advanced Grid services (resource management, security, resource discovery) while developing higher-level components in terms of the familiar and powerful application development frameworks provided by commodity technologies. In each case, we also face the challenge of developing appropriate interfaces between Grid and commodity concepts and technologies—and, if similar Grid and commodity services are provided, reconciling competing approaches.

As part of these activities, we have successfully developed a Java-based Commodity Grid Toolkit (Java CoG Kit) that defines and implements a set of general components mapping Grid functionality into the Java framework. The Java CoG Kit is of particular interest for PSE developers because it allows them to implement preinstalled heavy-weight applications to be started on user-accessible com-

pute servers, as well as lightweight Web interfaces or portals allowing access to sophisticated remote compute services.

The primary goal of our research is not to build a PSE that will solve a specific problem for a particular application area. Instead, our focus is on developing a software infrastructure to make it easier to build and deploy powerful PSEs. We have based our development of the Java CoG kit on our experiences with application users in various problem domains. Thus, we are confident that the toolkit is general enough to be useful for a large number of PSE developers.

While we have introduced in [17] the general concepts of the Java CoG Kit, we will illustrate in this paper its practical use in the development of problem solving environments. Additionally, we introduce here new components and more sophisticated security concepts that are of particular interest to developers of chemistry problem-solving environments.

2. Portals to Problem Solving Environments

For readers to understand the scope of this work, we explain the terms *problem solving environment* and *portal*, since multiple definitions are used for both terms in the literature.

2.1. Problem Solving Environment

Our understanding of a PSE follows approximately the definition given in [7]: "A problem solving environment is a computational system that provides a complete and convenient set of high level tools for solving problems from a specific domain. The PSE allows users to define and modify problems, choose solution strategies, interact with and manage appropriate hardware and software resources, visualize and analyze results, and record and coordinate extended problem solving tasks. A user communicates with a PSE in the language of the problem, not in the language of a particular operating system, programming language, or network protocol."

For our research focus we assume that the problems must access remote resources, potentially in a secure fashion, and may require a large amount of compute and/or data resources. The process of solving the problem is steered by the scientist, and its progress may be monitored through Internet browsers or special-purpose application-monitoring programs.

2.2. Requirements for PSE Portals

We identified a list of characteristics that influenced our PSE toolkit design [1]:

Problem-oriented. The PSE should allow specialists to concentrate on their discipline, without having to become

experts in computer science issues, such as networks, parallel computing, or the World Wide Web.

Integrated. Many problems and their solution strategies are extremely heterogeneous: in models, codes, applications, and machines. A PSE must be designed to manage this heterogeneity in an integrated way, so that the user is presented with a predictable and consistent PSE.

Collaborative. Most science and engineering projects are performed in collaborative mode with physically distributed participants. A PSE must include the ability to foster collaborative solution strategies. We assume that a general-purpose video conferencing tool can be provided with common off-the-shelf tools developed by commercial companies. Nevertheless, it may be necessary to develop special-purpose collaborative tools that are not provided by third parties.

Distributed. Besides the need to support distributed collaboration between scientists, many problems we have been dealing with (such as Grand Challenges) can be solved only while accessing large distributed resources (such as storage and compute resources) in conjunction with each other. A PSE must be able to access these distributed resources seamlessly and in collaboration.

Persistent. Since developing a solution for a problem may require significant time, it is desirable to provide a persistent environment that allows the researcher to resume the solution process at a later time at a potentially different location. Thus, it is necessary to be able to checkpoint not only the state of the calculation but also the state of the PSE user interface. The persistence of a PSE could be enhanced with preferences that are either set by the user or are detected automatically by the PSE. Such functionality could be achieved with the integration of what is called an electronic notebook.

Open, flexible, adaptive. Problem strategies require being able to integrate novel ideas. A sophisticated PSE building tool must be able to tailor or add new functionality within its existent base.

Graphical, visual. The use of graphics and visuals can enhance the usability of the PSE, for example, through animated tables and directed graphs to visualize the state of the application. Furthermore, it must be possible to integrate custom-designed graphical and visual inputs and outputs.

2.3. Portal for Problem Solving Environments

A "Web portal" is commonly defined as an entry point or starting site for the World Wide Web, combining a mixture of content and services that attempts to provide a personalized "home base" for its audience. Features include customizable start pages to guide users easily through the services provided by the portal. Such services include filterable e-mail, chat rooms and message boards, person-

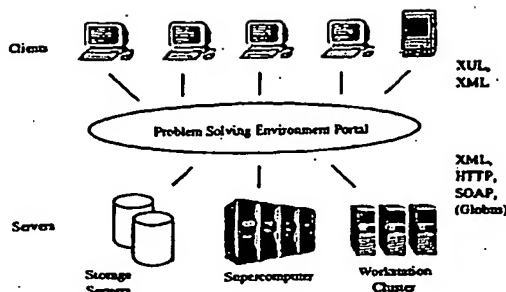


Figure 1. A computing portal interfaces clients with Grid resources such as storage servers, supercomputers, and workstation clusters.

alized news, gaming channels, shopping capabilities, advanced search engines, and personal homepage construction kits. Examples for consumer-oriented portals are provided by AOL and Yahoo.

In this spirit, we suggest that a convenient way of interfacing with a PSE is to design portals for a scientific domain or a particular problem strategy. Besides providing collaborative, interactive, and information services, such portals include also services that are unique for the domain but are typically not provided by consumer-oriented portals. These services include interfaces between users of the PSE with the help of clients ranging from graphics workstations to palm pilots to the resources available as part of the computational Grid (Figure 1). Naturally, not all capabilities of a portal may be exposed by less capable access devices such as palm pilots. Nevertheless, the ability to send a message to a beeper, palm pilot, or cell phone adds significant value to the PSE functionality by notifying the user of the existence of a collaborative session or the completion of a problem solution. Hence, the ability to access a portal with various (even less capable) devices is an integral part of our design.

2.4. Users and Usage Modes of PSE Portals

Portal development for PSEs first requires determining which customer group will be using the portal. We distinguish three target groups:

1. *Novice* science or problem solving environment users, that is, casual or novice users using readily available solutions to problems. The problem strategy is non-transparent to novice users.
2. *Expert* science or problem solving environment users, that is, users in the domain for which the portal is

developed. Such users are able to extend the portal while providing solution strategies as used by the novice users or themselves.

3. *Developer* of application or problem solving environments, providing general-purpose components used by experts or novice users.

In addition, we distinguish between interactive and batch mode in which jobs are submitted from the problem solving environment to the backend systems by the users. We have to be able to support the use of compute resources through fine-grained parallel programs, typically provided through MPI message-passing parallel programs, or coarse-grain parallel programs through job dependencies between jobs submitted to the batch processing systems or a fork jobmanager. The toolkit we describe in this paper supports these usage modes.

3. Architecture

Because of the diversified use of a PSE portal, the architecture of such an environment must be flexible. Thus, it is not feasible to develop a point solution for a single problem. Needed instead is a portal toolkit that includes a set of services exposed via APIs that can be used to assemble a point solution for a problem. Figure 2 and Table 1 outline the various groups of services that we initially focus on and that must be integrated into a portal toolkit. Each portal component may have several subcomponents that support the tasks performed as part of the computing portal for problem solving environments. The components in bold text of Figure 2 are developed as part of the CoG Kit. Other components are provided either by commodity software or the application programmers. The flexible design makes it possible to integrate new components into the framework or replace existing modules.

3.1. Grid Core Services

The scientific problem-solving infrastructure of the twenty-first century will support the coordinated use of numerous distributed heterogeneous components, including advanced networks, computers, storage devices, display devices, and scientific instruments. The term "The Grid" is often used to refer to this emerging infrastructure [5][6]. NASA's Information Power Grid and the NCSA Alliance's National Technology Grid are two contemporary projects prototyping Grid systems; both build on a range of technologies, including many provided by the Globus project in which we are involved. In designing PSE portals, we make extensive use of these technologies, including Globus services, such as

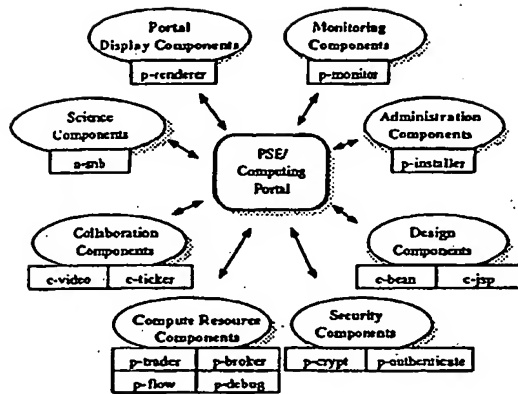


Figure 2. A computing portal is built with the help of a variety of portal components ranging from specialized application-specific portal components to components for using distributed compute resources or other Grid infrastructure.

Table 1: Portal Components

Portal Component	Sub Component	Function
Collaboration	c-video	Video collaboration (e.g. netmeeting)
	c-ticker	news server
Design	p-bean	Java IDE (e.g. VisualJava, Forte, ...)
	p-jsp	Java IDE
Science	a-smb	Application specific provided by scientists
	p-trader	locates compute resources
Compute Resource	p-broker	schedules jobs
	p-flow	dependencies between jobs
	p-debug	debugs job execution
	gram	Globus job submission
Security	p-crypt	sends secure messages
	p-authenticate	authenticates to the system
	gsi	Grid Security Infrastructure
Administration	p-installer	installs software on client
Monitoring	p-monitor	monitors the state
	mds	Globus Metacomputing Directory Service
Display	p-renderer	displays information from XML

- the information service (MDS), which enables uniform access to information about the structure and state of Grid resources;
- an authentication and authorization service (GSI), which provides mechanisms for establishing, identifying, and creating delegatable credentials; and
- a uniform job submission service across distributed scheduling systems (GRAM).

These Grid services are often termed "middleware": they typically involve a distributed state and can be viewed as a natural evolution of the services provided by today's Internet. They build the basis for developing a Grid-based problem solving environment because many of the portal components use their services.

3.2. Job Submission and Execution

One of the main services a PSE portal must provide is to job submission to remote resources. This must be done in seamless fashion from the desktop with a single sign-on authentication. Computers must be located and the computation must be started on the selected systems. It is essential to monitor the progress of the job execution and obtain the results of the calculation through, for example, output files that may be manipulated locally on the client side (the computer from which the job was initiated). We are able to support such uniform job submission while using the Globus metacomputing toolkit to access Grid resources securely.

Authentication The first step of the job submission is to authenticate with the system. Authentication is the process to verify the identity of an entity. Although the cryptographic algorithms that form the basis of most security systems—such as public key cryptography—are relatively simple, it is a challenging task to use these algorithms to meet diverse security goals in complex, dynamic problem solving environments, with potentially large and dynamic sets of users and resources and fluid relationships between users and resources. Authentication solutions for problem solving environments in a Computational Grids must solve two problems not commonly addressed by standard authentication technologies.

The first problem is support for local heterogeneity. The resources available in the Grid are operated by a diverse range of entities, each defining a different administrative domain.

The second problem support for N-way security contexts. In traditional client-server applications, authentication involves just a single client and a single server. In contrast, a Grid-based PSE may require and dynamically

maintained resources. Thus, it must be possible to establish a security relationship between any two processes in the computation used to solve the problem even if they are in different administrative domains. To simplify our task we use the Grid security infrastructure (GSI) that deals with the authentication. GSI policy allows a user to authenticate just once per computation, at which time a credential is generated that allows processes created on behalf of the user to acquire resources, and so on, without additional user intervention. Local heterogeneity is handled by mapping a user's Grid identity into local user identities at each resource. In summary, the GSI security model provides PSEs the following advantages: single sign-on for all resources, no need for user to keep track of accounts and passwords at multiple sites, and no plaintext passwords.

Protocol-based Job Submission Recently, Globus has been enhanced to include an HTTP-based protocol for job submission. Thus, job submission can be initiated from a client on which no other Globus components are installed. Figure 3 shows the Globus components that are involved in such a job submission. First, one has to authenticate with the system, which is done with the help of public key infrastructure and a proxy delegation while generating a temporary key. Jobs are submitted from the client side through API calls known as *gram-submit* and *gram-request*. The gatekeeper on the Globus-enabled resource verifies whether the user is allowed to submit a job to it and checks the availability of the user's public key in a grid map file local to the resource. Once a job has been successfully submitted to the system, it is started with the help of the job manager, and its state is monitored with the help of the reporter. During startup of a job a user can register callback handlers that provide job status updates. In our Java CoG Kit we have implemented all components and services responsible for the proxy initialization and the job submission. Furthermore we have replaced the C-based callback service with a Java-based event service. Thus, all components to submit a job are available in pure Java, allowing even Windows clients to submit jobs to Globus servers.

3.3. Additional Security Issues

In the preceding sections we addressed security issues related to authentication and authorization while using the security policy suggested by Globus. The authorization to use a particular Grid resource can be controlled via a grid-map file and appropriately specified group permissions controlled by the local system administrators.

Nevertheless, we still have to address issues such as the secure communication between programs. To guarantee privacy, we use the security mechanisms provided by secure socket connections, which we can obtain through GlobusIO.

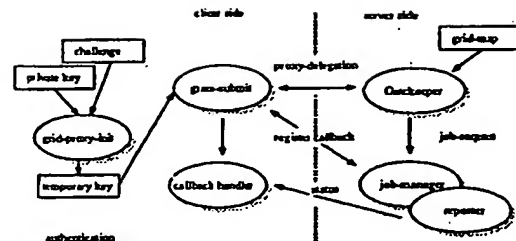


Figure 3. The components of the Globus security infrastructure used during job submission. All client side components are available within the CoG Kit as pure Java components.

This allows us to send messages and data in a secure fashion between compute resources.

4. Java CoG Kit

In the remainder of this paper we focus our attention on our Java CoG Kit prototype, which enables us to build the components listed in Table 1 and used as part of a PSE. Because of the large number of packages and classes required to expose the necessary functionality of the Globus toolkit, we focus in this paper on a subset of the classes that we deem most useful for the development of PSE-based Grid applications. The design of the Java CoG Kit is intended to facilitate the development of future components as a community project. To support an iterative process of definition, development, and application of a Java CoG Kit in collaboration with other teams, we classify components in four layers. This categorization provides the necessary subdivisions to coordinate such a challenging open community software engineering task.

Low-Level Grid Interface Components provide mappings to commonly used Grid services: for example, the *Grid information service* (the Globus Metacomputing Directory Service, MDS), which provides Lightweight Directory Access Protocol (LDAP) [9] access to information about the structure and state of Grid resources and services; *resource management services*, which support the allocation and management of computational and other resources (via the Globus GRAM and DUROC services); and *data access services*, for example, via the Globus GASS service [3].

Low-Level Utility Components are utility functions designed to be reused by many users. Examples are com-

```

// Step 0. Initialization
MDS mds=new MDS("www.globus.org", "389", "o=Grid");
// Step 1. Search for an available machine
result = mds.search
("(<i>objectclass=GridComputeResource</i>(freednodes=64))",
 "context");
// Step 1.a) Select a machine
machineContact = <i>select the machine with minimal
execution time from the contacts that are returned in result</i>
// Step 2. Prepare the data for the experiment
// Step 2.a) Search for the data and return
// the attributes: server,port,directory,file
dn = mds.search
("(<i>objectclass=MoleculeStructureData</i>(name=cholera)",
 "dn", MDS.SubtreeScope);
result = mds.lookup(dn, "server,port,directory,file");
// Step 2.b) download the data to the machine
url = result.get("server")+"."+result.get("port")+"."
+result.get("directory")+"."+result.get("file");
data = server.fetch(url, machineContact);
// Step 3. Prepare a description for running the model
RSL rsl = new RSL("(<i>executable=snb</i>(processors=64)
(arguments=-out snb.out)
(arguments=-in " + data.filename +"))");
// Step 4. Submit the program
GramJob job = new GramJob();
job.addListener(new GramJobListener() {
    public void stateChanged(GramJob job) {
        // react to job state changes
    }
});
try {
    job.request(machineContact, rsl);
} catch (GramException e) {
    // problem submitting the job
}

```

Figure 4. This sample script demonstrates how we access basic Grid services with the help of the Java CoG Kit. Here data for a structural biology code called SnB are located, an appropriate machine is selected, and the calculation is executed on that machine.

ponents that use information service functions to find all compute resources that a user can submit to, that prepare and validate a job specification while using the extended markup language (XML) or the Globus job submission language (RSL), that locate the geographical coordinates of a compute resource and that test whether a machine is alive.

Low-Level GUI Components provide a basic graphical components that can be reused by application developers. Examples are LDAP attribute editors, RSL editors, LDAP browsers, and search components.

Application-specific GUI Components simplify the bridge between applications and the basic CoG Kit components. Examples are a stock market monitor, a graphical climate data display component, or a specialized search engine for climate data.

Figure 4 shows how a small set of services provided by the Java CoG Kit may be used in practice. This Java program skeleton demonstrates how simple it is to build portal-specific services when accessing a variety of basic Grid services through the Java CoG Kit. In this example, an appropriate machine is selected for execution, data for an instantiation of a problem specific algorithm is determined, and the job is executed on that machine, resulting in the generation of an output file.

4.1. Low-Level Grid Interface Components

We describe here a subset of packages that provide the interface to the low-level Grid services and application interfaces. These packages are used by many users to develop Java-based programs in the Grid. We describe only the general functionality of these packages. A complete list of the classes and methods accompanies the distribution [18].

RSL The package *org.globus.rsl* provides methods for creating, manipulating, and checking the validity of the Resource Specification Language (RSL) expressions used in Globus [8] to express resource requirements. As shown in Step 3 of Figure 4, the arguments to a *new* call include parameters that specify both characteristics of the required resources and properties of the computation.

GRAM The package *org.globus.gram* provides a mapping to the Globus Resource Allocation Manager (GRAM) services [8], which allow users to schedule and manage remote computations. The classes and methods distributed allow users to submit jobs, bind to already submitted jobs, and cancel jobs on remote computers. Other methods allow users to determine whether they can submit jobs to a specific resource (through a Globus gatekeeper) and to monitor the job status (*pending*, *active*, *failed*, *done*, and *suspended*).

As shown in Step 4 of Figure 4 the class *Gram* is used to create a job with an RSL string describing the job and a machine contact that determines on which machine the job is requested for execution. Our Java mapping differs from that provided in Globus for C through the introduction of a formal job object, as well as the availability of a sophisticated event model in Java. Our implementation utilizes this event model and transfers the C callbacks into equivalent Java events. In Java one can now use threads in order to "listen" to a particular event that can trigger further actions. A Java interface *GramJobListener* that contains the method *stateChanged(GramJob job)* can be used to define customized job listeners that can be added with the *GramJob* method *addListener(GramJobListener listener)*.

MDS The package *org.globus.mds* simplifies access to the Metacomputing Directory Service (MDS) [15], which is an important part of the Globus information service. Its functions include (a) establishing a connection to an MDS server, (b) querying MDS contents, (c) printing, and (d) disconnecting from the MDS server. The package provides an intermediate application layer that can be easily adapted to different LDAP [9] client libraries, including JNDI [10], Netscape SDK [11], and Microsoft SDK [13].

As shown in Step 1 of Figure 4, the parameters to initialize the MDS class are the DNS name of the MDS server, the port number for the connection, and the distinguished name (DN) that specifies the root for a search in the directory tree. A search is performed in Step 2a; the first parameter specifies the top level of the tree in which the search is performed, the second parameter specifies the LDAP query, and the third parameter specifies the scope, that is, for how many levels in the tree the search should continue (in our case, only the next level). Search results can also be stored in a NamingEnumeration provided by JNDI.

GASS The Global Access to Secondary Storage (GASS) service [3] simplifies the porting and running of applications that use file I/O, eliminating the need to manually log onto sites and ftp files or to install a distributed file system. The package *org.globus.gass* provides an essential subset of GASS services to support the copying of files between computers on which the Grid Services are installed. The method *get(String from, String to)* copies a remote file to a local file, and the method *put(String from, String to)* copies a local file to a remote location. The *fetch* method used in our example (Figure 4) provides a convenient wrapper and uses internally the previously mentioned *get* method.

4.2. Low-Level Utilities

The low-level utility classes currently defined in the CoG Kit provide an abstract datatype representing acyclic graphs and basic XML parsing routines. The graph class is used, for example, to access dependencies between jobs, a major requirement for PSEs. The XML classes are used to provide transformations between different data formats. Using XML has the advantage that a Document Type Definition (DTD) that is defined for these data formats can be used to verify whether a record to be transmitted is well formed before it is sent to a server. Thus the load on servers can be dramatically reduced. The availability of a dependency between jobs is a significant extension to the existing Globus low-level application interface. In addition, we have defined a general concept of a *machine* and *job broker* interface. This enables a programmer to define a customized selection of machines and jobs dependent on his demand. We have used this technology as part of a high-throughput broker

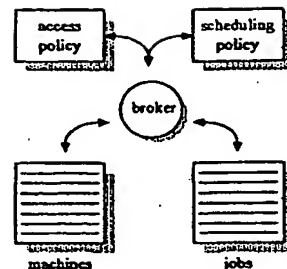


Figure 5. A broker interface allows us to specify an easy way to develop compatible components relying on this interface. Jobs and machines are selected based on a pre-defined access/security policy as well as a scheduling policy. The policies may be generated dynamically based on other system information.

that is implemented in Java but can also be exposed through CORBA objects. The GECCO application introduced in Section 4.4 uses the Java-based machine and job brokers.

The broker is a good example of a universally useful component for PSE developers, as well as Grid users. Here a set of jobs and machines is stored in two tables. Dependent on a scheduling and access policy, a machine is selected and a job is scheduled for the execution on this machine (see Figure 5). We have defined a simple interface outlined in Figure 6. This interface allows us to add jobs and machines to the sets so that it is possible to administer them dynamically. With the help of this interface we have defined multiple scheduling policies such as first-come-first-served and load balancing based on resource characteristics. Currently we are investigating the use of economy models for scheduling jobs to machines.

4.3. Low-Level GUI Components

The Java CoG Kit low-level GUI components provide basic graphical components that can be used to build more advanced GUI-based applications. These components include text panels that format RSL strings, tables that display results of MDS search queries [17], trees that display the directory information tree of the MDS, and tables to display HBM and network performance data. Each component can be customized and is available as JavaBean. In future releases of the Java CoG Kit it will be possible to integrate the bean in a Java-based GUI composition tool such as JBuilder or VisualCafe.

```

interface broker ... {
    addJob(JobDescription job)
    deleteJob(JobDescription job)
    addMachine(MachineDescription machine)
    deleteMachine(MachineDescription machine)
    setAccessPolicy(BrokerAccessPolicy policy)
    setSchedulingPolicy(BrokerSchedulingPolicy policy)

    ...

    MachineDescription getMachine()
    JobDescription getJob()

    ...
}

```

Figure 6. This code fragment shows the elementary methods of the broker. Jobs and machines can be added. The job and machine returned by the get methods are defined by the policies and the algorithms defined by an object instantiation of the interface.

4.4. PSE Application Level Utilities and GUI Components

High-level graphical applications combine a variety of CoG Kit components to deliver a single application or applet. These applications can be combined to provide even greater functionality. The user should select the tools that seem appropriate for the task. To demonstrate the range of applications, we have included a set of screen dumps that highlight the look and feel of some applications developed to date.

GECCO The Graph Enabled Console Component (GECCO) is a graphical tool for specifying and monitoring the execution of sets of tasks with dependencies between them [16][14]. Specifically it allows one to

1. specify the jobs and their dependencies graphically or with the help of an XML-based configuration file;
2. debug the specification in order to find erroneous specification strings before the job is submitted; and
3. execute and monitor the job graphically and with the help of a log file.

As shown in Figure 7, each job is represented as a node in the graph. A job is executed as soon as its predecessors are reported to have successfully completed. The state of a job is animated with colors. It is possible to modify the specification of the job while clicking on the node: A specification window pops up allowing the user to edit the RSL, the label, and other parameters. Editing can also be performed during runtime (job execution), hence providing for simple computational steering.

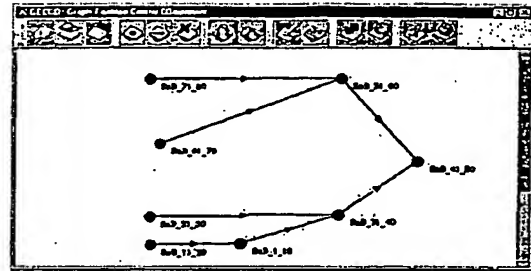


Figure 7. The Grid Enabled Console Component (GECCO) allows the user to specify dependencies between tasks that are to be executed in the Grid environment.

High-Throughput Broker We have developed a prototype of a high-throughput broker to test whether the interfaces and classes allow one to easily generate high-level components that simplify job maintenance tasks for certain problem-solving strategies. One of the tasks that has been identified and is common to many solution strategies is to perform a parameter study [2][4]. That is, an algorithm is repeatedly executed with a variety of parameters. Our system is based on the interface of a broker and thus allows us to clearly separate the GUI presentation from the functionality (Figure 8). The prototype looks for compute resources available in a pool of machines formed by a Grid information service with the help of the Globus MDS. From this pool we select those resources that are idle and are available for calculation. If a resource is not able to fulfill a job (because of connection timeout or excessive time needed to complete the job), the resource is automatically removed from the set of viable candidates. The set of resources as well as those removed from the list can be manipulated through an interactive shell. A similar interface exists for the jobs. Special attention has to be placed on the implementation of such a broker. Although it is possible to spawn for each job and machine a thread that maintains the appropriate object, we have chosen to maintain the jobs and machines in lists to avoid the overhead associated with threads and the expected resource limitations on the machine on which the system is running. Thus, we are able to handle submissions that maintain 10,000 or more jobs, a task that would otherwise be impossible.

5. Installation and Upgrading

An important function that must be provided by a PSE is to install and upgrade the software that accesses the various services exposed as part of its design. Using Java will provide us with several options for deploying our client soft-

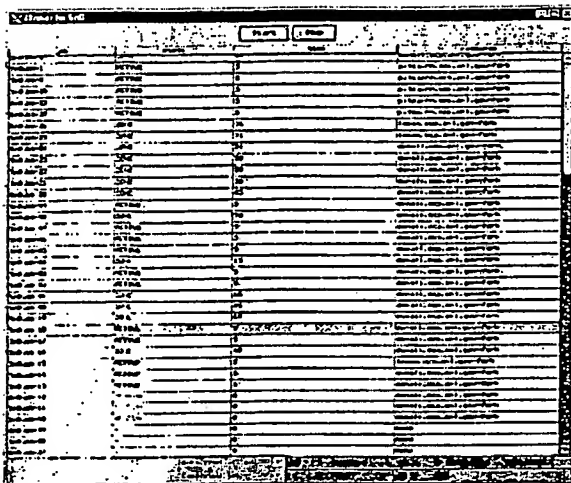


Figure 8. A high throughput broker allows the submission of many jobs as part of a problem. After all jobs are completed a solution of the problem can be obtained. The progress of the calculation is monitored with a GUI.

ware. In addition to traditional methods of delivering client software to be installed and configured prior to its use, we can develop thin-client software, which can be dynamically installed or updated as well as loaded at time of use.

Preinstallation of the software in the form of a standalone application or a library is convenient for applications that would take too long to be installed via a network connection (Figure 9). This strategy is today used by many commercial portals as part of their access software enabled with the help of so-called browser plug-ins. Nevertheless, we recognize the fact that it is sometimes not possible to install any software on the client computer because the user does not have sufficient access to it. This requires, at the cost of additional download time, downloading the appropriate *jar* files from a well-defined URL. In both cases it will be possible to augment the *jar* files with authentication measures in the form of certificates. These will allow clients to identify the source of the code upon downloading our software and to verify that it can be trusted for use on their systems.

6. Summary

Commodity distributed-computing technologies enable the rapid construction of sophisticated client-server applications. Grid technologies provide advanced network ser-

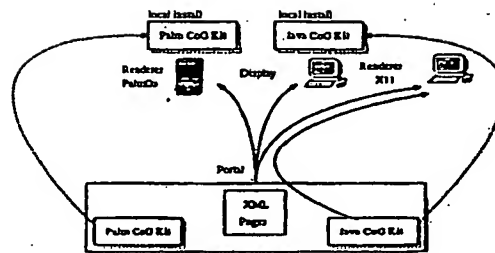


Figure 9. The installation of the CoG Kit onto a client can be done prior to the start of the application as a standalone application or the installation of a library or during an on-demand execution.

vices for large-scale, wide area, multi-institutional environments and for applications that require the coordinated use of multiple resources. In the Commodity Grid project, we seek to bridge these two worlds so as to enable advanced applications that can benefit from both Grid services and sophisticated commodity development environments.

The Java Commodity Grid Toolkit (CoG Kit) described in this paper represents a first attempt at creating of such a bridge. Building on experience gained over the past three years with the use of Java in Grid environments, we have defined a set of classes that provide the Java programmer with access to basic Grid services, enhanced services suitable for the definition of desktop problem solving environments, and a range of GUI elements. Initial experiences with these components have been positive. It has proven possible to recast major Grid services in Java terms without compromising on functionality. Some substantial Java CoG Kit applications have been developed, and reactions from users have been positive.

Our future work will involve the integration of more advanced services into the Java CoG Kit and the creation of other CoG Kits, with CORBA, DCOM, and Python being early priorities. We also hope to gain a better understanding of where changes to commodity or Grid technologies can facilitate interoperability and of where commodity technologies can be exploited in Grid environments.

With the help of the CoG Kits we have prototyped a portal to a structural biology problem solving environment. Other projects are currently investigating the use of the CoG Kit to simplify the access to Grid resources. Such projects include the astrophysics portal Cactus, the NCSA Userportal, and SDSC Hotpage. The requirements demanded by such projects have influenced our present design, and we are collaborating with project developers to enhance the components we provide in the CoG Kit. Most recently, we have

started to address the integration of components developed by other collaborators.

7. Acknowledgments

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Globus research and development is supported by DARPA, DOE, and NSF. We thank Geoffrey C. Fox, Dennis Gannon, and Jason Novotny for valuable discussions during the course of the CoG Kit development. This work would not have been possible without the help of the Globus team.

For up-to-date release notes, and further information readers should refer to the Web page <http://www.globus.org/cog> [18].

References

- [1] Marc Abrams, Donald Allison, Dennis Kafura, Calvin Ribbens, Mary Beth Rosson, Clifford Shaffer, and Layne Watson. PSE Research at Virginia Tech: An Overview. Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, 1999.
- [2] D. Abramson, R. Sasic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc. 4th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1995.
- [3] Joseph. Bester, Ian Foster, Carl Kesselman, Jean Tedesco, and Steven Tuecke. GASS: A data movement and access service for wide area computing systems. In *Proc. IOPADS'99*. ACM Press, 1999.
- [4] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A Worldwide Flock of Condors: Load Sharing among Workstation Clusters. *Future Generation Computer Systems*, 12, 1996.
- [5] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan-Kaufmann, 1999.
- [6] Ian Foster. Building the Grid: An Integrated Services and Toolkit Architecture for Next Generation Networked Applications. http://www.gridforum.org/building_the_grid.htm, July 1999.
- [7] E. Gallopoulos, E. Houstis, and J.R. Rice. Problem-Solving Environments for Computational Science. *IEEE Computational Science and Engineering*, 1:11-23, 1994.
- [8] The Globus GRAM. <http://www.globus.org/gram>.
- [9] Tim. Howes and Mark Smith. *LDAP : Programming Directory-Enabled Applications With Lightweight Directory Access Protocol*. Technology Series. Macmillan Technical Publishing, 1997.
- [10] JAVA Naming and Directory Interface (JNDI). <http://java.sun.com/products/jndi>. Version 1.2.
- [11] Netscape Directory and LDAP Developer Central. <http://developer.netscape.com/tech/directory/index.html>.
- [12] J. R. Rice and R. F. Boisvert. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering*, Fall:44-53, 1996.
- [13] Richard Schwartz. *Windows 2000 : Active Directory Survival Guide*. John Wiley and Sons, 1999.
- [14] Gregor von Laszewski. A Loosely Coupled Metacomputer: Cooperating Job Submissions across Multiple Supercomputing Sites. *Concurrency, Experience, and Practice*, Mar. 2000.
- [15] Gregor von Laszewski, S. Fitzgerald, I. Foster, C. Kesselman, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proc. 6th IEEE Symp. on High-Performance Distributed Computing*, pages 365-375, 1997.
- [16] Gregor von Laszewski and Ian Foster. Grid Infrastructure to Support Science Portals for Large Scale Instruments. In *Proc. of the Workshop Distributed Computing on the Web (DCW)*. University of Rostock, Germany, June 1999.
- [17] Gregor von Laszewski, Ian Foster, Jarek Gawor, Warren Smith, and Steve Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM 2000 Java Grande Conference*, San Francisco, California, June 3-4, 2000. <http://www.extreme.indiana.edu/java00>.
- [18] Gregor von Laszewski, Jarek Gawor, and Peter Lane. Java CoG Distribution. <http://www.globus.org/cog>, January 2000. Version 0.8.6.

Grid-based Asynchronous Migration of Execution Context in Java Virtual Machines

Gregor von Laszewski¹, Kazuyuki Shudo², and Yoichi Muraoka²

¹ Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL, U.S.A.
gregor@mcs.anl.gov

² School of Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan {shudoh, muraoka}@muraoka.info.waseda.ac.jp

Abstract. Previous research efforts for building thread migration systems have concentrated on the development of frameworks dealing with a small local environment controlled by a single user. Computational Grids provide the opportunity to utilize a large-scale environment controlled over different organizational boundaries. Using this class of large-scale computational resources as part of a thread migration system provides a significant challenge previously not addressed by this community. In this paper we present a framework that integrates Grid services to enhance the functionality of a thread migration system. To accommodate future Grid services, the design of the framework is both flexible and extensible. Currently, our thread migration system contains Grid services for authentication, registration, lookup, and automatic software installation. In the context of distributed applications executed on a Grid-based infrastructure, the asynchronous migration of an execution context can help solve problems such as remote execution, load balancing, and the development of mobile agents. Our prototype is based on the migration of Java threads, allowing asynchronous and heterogeneous migration of the execution context of the running code.

1 Introduction

Emerging national-scale *Computational Grid* infrastructures are deploying advanced services beyond those taken for granted in today's Internet, for example, authentication, remote access to computers, resource management, and directory services. The availability of these services represents both an opportunity and a challenge: an opportunity because they enable access to remote resources in new ways, a challenge: because the developer of thread migration systems may need to address implementation issues or even modify existing systems designs. The scientific problem-solving infrastructure of the twenty-first century will support the coordinated use of numerous distributed heterogeneous components, including advanced networks, computers, storage devices, display devices, and scientific instruments. The term *The Grid* is often used to refer to this emerging infrastructure [5]. NASA's Information Power Grid and the NCSA Alliance's National Technology Grid are two contemporary projects prototyping Grid systems; both build on a range of technologies, including many provided by the Globus project. Globus is a metacomputing toolkit that provides basic services for security, job submission, information, and communication.

The availability of a national Grid provides the ability to exploit this infrastructure with the next generation of parallel programs. Such programs will include mobile code as an essential tool for allowing such access enabled through *mobile agents*. Mobile agents are programs that can migrate between hosts in a network (or Grid), in order to find places of their own choosing. An essential part for developing mobile agent systems is to save the state of the running program before it is transported to the new host, and restored, allowing the program to continue where it left off. Mobile-agent systems differ from process-migration systems in that the agents move when they choose, typically through a *go* statement, whereas in a process-migration system the system decides when and where to move the running process (typically to balance CPU load) [9]. In an Internet-based environment mobile agents provide an effective choice for many applications as outlined in [11]. Furthermore, this applies also to Grid-based applications. Advantages include improvements in latency and bandwidth of client-server applications and reduction in vulnerability to network disconnection. Although not all Grid applications will need mobile agents, many other applications will find mobile agents an effective implementation technique for all or part of their tasks. The migration system we introduce in this paper is able to support mobile agents as well as process-migration systems, making it an ideal candidate for applications using migration based on the application as well as system requirements.

The rest of the paper is structured as follows. In the first part we introduce the thread migration system MOBA. In the second part we describe the extensions that allow the thread migration system to be used in a Grid-based environment. In the third part we present initial performance results with the MOBA system. We conclude the paper with a summary of lessons learned and a look at future activities.

2 The Thread Migration System MOBA

This paper describes the development of a Grid-based thread migration system. We based our prototype system on the thread migration system MOBA, although many of the services needed to implement such a framework can be used by other implementations.

The name *MOBA* is derived from *MOBile Agents*, since this system was initially applied to the context of mobile agents [17][22][14][15]. Nevertheless, MOBA can also be applied to other computer science-related problems such as the remote execution of jobs [4][8][3]. The advantages of MOBA are threefold:

1. **Support for asynchronous migration.** Thread migration can be carried out without the awareness of the running code. Thus, migration allows entities outside the migrating thread to initiate the migration. Examples for the use of asynchronous migration are global job schedulers that attempt to balance loads among machines. The program developer has the clear advantage that minimal changes to the original threaded code are necessary to include sophisticated migration strategies.
2. **Support for heterogeneous migration.** Thread migration in our system is allowed between MOBA processes executed on platforms with different operating systems. This feature makes it very attractive for use in a Grid-based environment, which is by nature built out of a large number of heterogeneous computing components.

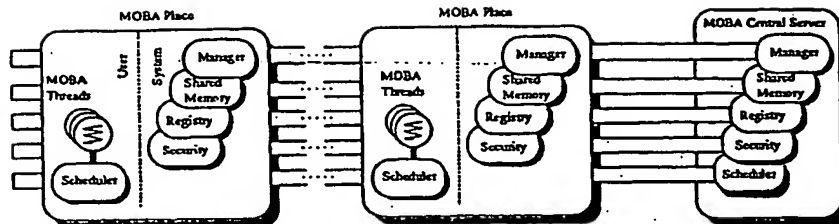


Fig. 1. The MOBA system components include MOBA places and a MOBA central server. Each component has a set of subcomponents that allow thread migration between MOBA places.

3. Support for the execution of native code as part of the migrating thread. While considering a thread migration system for Grid-based environments, it is advantageous to enable the execution of native code as part of the overall strategy to support a large and expensive code base, such as in scientific programming environments. MOBA will, in the near future, provide this capability. For more information on this subject we refer the interested reader to [17].

2.1 MOBA System Components

MOBA is based on a set of components that are illustrated in Figure 1. Next, we explain the functionality of the various components:

Place. Threads are created and executed in the *MOBA place* component. Here they receive external messages to move or decide on their own to move to a different place component. A MOBA place accesses a set of MOBA system components, such as manager, shared-memory, registry, and security. Each component has a unique functionality within the MOBA framework.

Manager. A single point of control is used to provide the control of startup and shut-down of the various component processes. The manager allows the user to get and set the environment for the respective processes.

Shared Memory: This component shares the data between threads.

Registry: The registry maintains necessary information — both static and dynamic — about all the MOBA components and the system resources. This information includes the OS name and version, installed software, machine attributes, and the load on the machines.

Security: The security component provides network-transparent programming interfaces for access control to all the MOBA components.

Scheduler: A MOBA place has access to user-defined components that handle the execution and scheduling of threads. The scheduling strategy can be provided through a custom policy developed by the user.

2.2 Programming Interface

We have designed the programming interface to MOBA on the principle of simplicity. One advantage in using MOBA is the availability of a user-friendly programming interface. For example, with only one statement, the programmer can instruct a thread to migrate; thus, only a few changes to the original code are necessary in order to augment an existent thread-based code to include thread migration. To enable movability of a thread, we instantiate a thread by using the `MobaThread` class instead of the normal Java `Thread` class. Specifically, the `MobaThread` class includes a method, called `goTo`, that allows the migration of a thread to another machine. In contrast to other mobile agent systems for Java [10][12][6], programmers using MOBA can enable thread migration with minor code modifications.

An important feature of MOBA is that migration can be ordered not only by the migrant but also by entities outside the migrant. Such entities include even threads that are running in the context of another user. In this case, the statement to migrate is included not in the migrant's code but in the thread that requests the move into its own execution context. To distinguish this action from the `goTo`, we have provided the method `moveTo`.

2.3 Implementation

MOBA is based on a specialized version of the Java Just-In-Time (JIT) interpreter. It is implemented as a plug-in to the Java Virtual Machine (JVM) provided by Sun Microsystems. Although MOBA is mostly written in Java, a small set of C functions enables efficient access to perform reflection and to obtain thread information such as the stack frames within the virtual machine. Currently, the system is supported on operating systems on which the Sun's JDK 1.1.x is ported. A port of MOBA based on JDK 1.2.x is currently under investigation. Our system allows heterogeneous migration [19] by handling the execution context in JVM rather than on a particular processor or in an operating system. Thus, threads in our system can migrate between JVMs on different platforms.

2.4 Organization of the Migration Facilities

To facilitate migration within our system, we designed MOBA as a layered architecture. The migration facilities of MOBA include introspection, object marshaling, thread externalization, and thread migration. Each of these facilities is supported and accessed through a library. The relationship and dependency of the migration facilities are depicted in Figure 2. The introspection library provides the same function as the reflection library that is part of the standard library of Java. Similarly, object marshaling provides the function of serialization, and thread externalization translates a state of the running thread to a byte stream.

The steps to translate a thread to a byte stream are summarized in Figure 3. In the first step, the attributes of the thread are translated. Such attributes include the name of the thread and thread priority. In the second step, all objects that are reachable from

the thread object are marshaled. Objects that are bound to file descriptors or other local resources are excluded from a migration. In the final step, the execution context is serialized. Since a context consists of contents of stack frames generated by a chain of method invocations, the externalizer follows the chain from older frames to newer ones and serializes the contents of the frames. A frame is located on the stack in a JVM and contains the state of a calling method. The state consists of a program counter, operands to the method, local variables, and elements on the stack, each of which is serialized in machine-independent form.

Together the facilities for externalizing threads and performing thread migration enabled us to design the components necessary for the MOBA system and to enhance the JIT compiler in order to allow asynchronous migration.

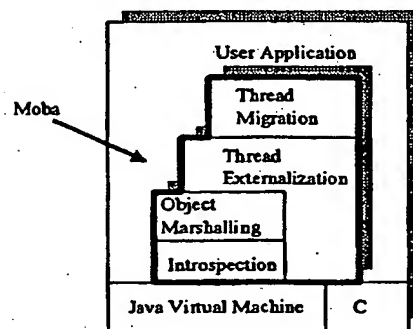


Fig. 2. Organization of MOBA thread migration facilities and their dependencies.

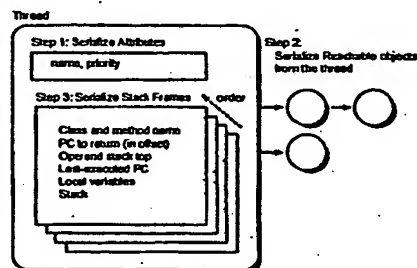


Fig. 3. Procedure to externalize a thread in MOBA.

2.5 Design Issues of Thread Migration in JVMs

In designing our thread migration system, we faced several challenges. Here we focus on five.

Nonpreemptive Scheduling. In order to enable the migration of the execution context, the migratory thread must be suspended at a *migration safe point*. Such migration safe points are defined within the execution of the JVM whenever it is in a consistent state. Furthermore, asynchronous migration within the MOBA system requires nonpreemptive scheduling of Java threads to prevent threads from being suspended at a not-safe point. Depending on the underlying (preemptive or nonpreemptive) thread scheduling system used in the JVM, MOBA supports either asynchronous or cooperative migration (that is, the migratory thread determines itself the destination). The availability of green threads will allow us to provide asynchronous migration.

Native Code Support. Most JVMs have a JIT runtime compiler that translates bytecode to the processors native code at runtime. To enable heterogeneous migration, a machine-independent representation of execution context is required. Unfortunately, most existing JIT compilers do not preserve a program counter on bytecode which is needed to reach a migration safe point. Only the program counter of the native code execution can be obtained by an existing JIT compiler. Fortunately, Sun's HotSpot VM [18] allows the execution context on bytecode to be captured during the execution of the generated native code since capturing the program counter on bytecode is also used for its dynamic deoptimization.

We are developing an enhanced version of the JIT compiler that checks, during the execution of native code, a flag indicating whether the request for capturing the context can be performed. This polling may have some cost in terms of performance, but we expect any decrease in performance to be small.

Selective Migration. In the most primitive migration system all objects reachable from the thread object are marshaled and replicated on the destination of the migration. This approach may cause problems related to limitations occurring during the access of system resources as documented in [17]. Selective migration may be able to overcome these problems, but the implementation is challenging because we must develop an algorithm determining the objects to be transferred. Additionally, the migration system must cooperate with a distributed object system, enabling remote reference and remote operation. Specifically, since the migrated thread must allow access to the remaining objects within the distributed object system, it must be tightly integrated within the JVM. It must allow the interchange of a local references and a remote references to support remote array access, field access, transparent replacement of a local object with a remote object, and so forth. Since no distributed object system implemented in Java (for example, Java RMI, Voyager, HORB, and many implementations of CORBA) satisfies these requirements, we have developed a distributed object system supported by the JIT compiler shuJIT [16] to provide these capabilities.

Marshaling Objects Tied to the Local Resource. A common problem in object migration systems is how to maintain objects that have some relation to resources specific to, say, a machine. Since MOBA does not allow to access objects that reside in a remote machine directly, it must copy or migrate the objects to the MOBA place issuing the request. Objects that depend on local resources (such a file and socket descriptors) are not moved within MOBA, but remain at the original fixed location [8][13].

Types of Values on the JVM Stack In order to migrate an object from one machine to another, it is important to determine the type of the local object variables. Unfortunately, Sun's JVM does not provide a type stack operating in parallel to the value stack, such as the Sumatra interpreter [1]. Local variables and operands of the called method stay on the stack. The values may be 32-bit or 64-bit immediate values or references to objects. It is difficult to distinguish the types only by their values.

With a JVM like Sun's, we have either to infer the type from the value or to determine the type by a data flow analysis that traces the bytecode of the method (like a bytecode verifier). Since tracing bytecode to determine types is computationally expensive,

we developed a version of MOBA that infers the type from the value. Nevertheless, we recently determined that this capability is not sufficient to obtain a perfect inference and validation method. Thus, we are developing a modified JIT compiler that will provide stack frame maps [2] as part of Sun's ResearchVM.

3 Moba/G Service Requirements

The thread migration system MOBA introduced in the preceding sections is used as a basis for a Grid-enhanced version which we will call MOBA/G. Before we describe the MOBA/G system in more detail, we describe a simple Grid-enhanced scenario to outline our intentions for a Grid-based MOBA framework. First, we have to determine a subset of compute resources on which our MOBA system can be executed. To do so, we query the Globus Metacomputing Directory Service (MDS) while looking for compute resources on which Globus and the appropriate Java VM versions are installed and on which we have an account. Once we have identified a subset of all the machines returned by this query for the execution of the MOBA system, we transfer the necessary code base to the machine (if it is not already installed there). Then we start the MOBA places and register each MOBA place within the MDS. The communication between the MOBA places is performed in a secure fashion so that only the application user can decrypt the messages exchanged between them. A load-balancing algorithm is plugged into the running MOBA system that allows us to execute our thread-based program rapidly in the dynamically-maintained MOBA places. During the execution of our program we detect that a MOBA place is not responding. Since we have designed our program with check-pointing, we are able to start new MOBA places on underutilized resources and to restart the failed threads on them. Our MOBA application finishes and deregisters from the Grid environment.

To derive such a version, we have tried to ask ourselves several questions:

1. What existent Grid services can be used by MOBA to enhance its functionality?
2. What new Grid services are needed to provide a Grid-based MOBA system?
3. Are any technological or implementation issues preventing the integration?

To answer the first two questions, we identified that the following services will be needed to enhance the functionality of MOBA in a Grid-based environment:

Resource Location and Monitoring Services. A resource location service is used to determine possible compute nodes on which a MOBA place can be executed. A monitoring service is used to observe the state and status of the Grid environment to help in scheduling the threads in the Grid environment. A combination of Globus services can be used to implement them.

Authentication and Authorization Service. The existent security component in MOBA is based on a simple centralized maintenance based on user accounts and user groups known in a typical UNIX system. This security component is not strong enough to support the increased security requirements in a Grid-based environment. The Globus project, however, provides a sophisticated security infrastructure that

can be used by MOBA. Authentication can be achieved with the concept of public keys. This security infrastructure can be used to augment many of the MOBA components, such as shared memory and the scheduler.

Installation and Execution Service. Once a computational resource has been discovered, an installation service is used to install a MOBA place on it and to start the MOBA services. This is a significant enhancement to the original MOBA architecture as it allows the shift from a static to a dynamic pool of resources. Our intention is to extend a component in the Globus toolkit to meet the special needs of MOBA.

Secure Communication Service. Objects in MOBA are exchanged over the IIOP protocol. One possibility is to use commercial enhancements for the secure exchange of messages between different places. Another solution is to integrate the Globus security infrastructure. The Globus project has initiated an independent project investigating the development of a CORBA framework using a security enhanced version of IIOP.

The services above can be based on a set of existing Grid services provided by the Globus project (compare Table 1). For the integration of MOBA and Globus we need consider only those services and components that increase the functionality of MOBA within a Grid-based environment.

Table 1. The Globus services that are used to build the MOBA/G thread migration system within a Grid-based environment. Services that are not available in the initial MOBA system are indicated with •.

MOBA/G Service	Service	Globus Component
MOBA Place startup	Resource Management	GRAM
MOBA Object migration	Communication	GlobusIO
• Secure Communication, Authentication, Secure component startup	Security	GSI
MOBA registry	Information	MDS
• Monitoring	Health and Status	HBM, NWS
• Remote Installation, Data Replication	Remote Data Access	GASS

Before we explain in more detail the integration of each of the services into the MOBA system, we point out that many of the services are accessible in Java through the Java CoG Kit. The Java CoG Kit [20][21] not only allows access to the Globus services, but also provides the benefit of using the Java framework as the programming model. Thus, it is possible to cast the services as JavaBeans and to use the sophisticated event and thread models as used in the programs to support the MOBA/G implementation. The relationship between Globus, the Java CoG Kit, and MOBA/G is based on a layered architecture as depicted in Figure 4.

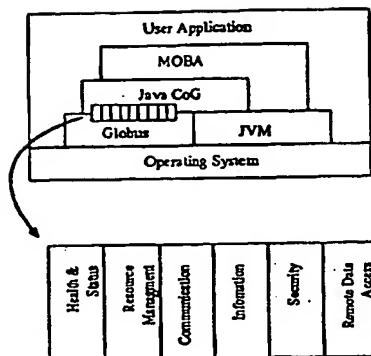


Fig. 4. The layered architecture of MOBA/G. The Java CoG Kit is used to access the various Globus Services.

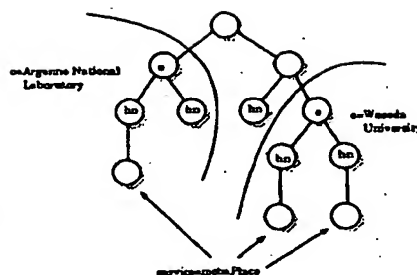


Fig. 5. The organizational directory tree of a distributed MOBA/G system between two organizations using three compute resources (hn) for running MOBA places.

3.1 Grid-based Registration Service

One of the problems a Grid-based application faces is to identify the resources on which the application is executed. The Metacomputing Directory Service enables Grid application developers and users to register their services with the MDS. The Grid-based information service could be used in several ways:

1. The existing MOBA central registry could register its existence within the MDS. Thus all MOBA services would still interact with the original MOBA service. The advantage of including the MOBA registry within the MDS is that multiple MOBA places could be started with multiple MOBA registries, and each of the places could easily locate the necessary information from the MDS in order to set up the communication with the appropriate MOBA registry.
2. The information that is usually contained within the MOBA registry could be stored as LDAP objects within the distributed MDS. Thus, the functionality of the original MOBA registry could be replaced with a distributed registry based on the MDS functionality.
3. The strategies introduced in (1) and (2) could be mixed while registering multiple enhanced MOBA registries. These enhanced registries would allow the exchange of information between each other and thus function in a distributed fashion.

Which of the methods introduced above is used depends on the application. Applications with high throughput demand but few MOBA places are sufficiently supported by the original MOBA registry. Applications that have a large number of MOBA places but do not have high demands on the throughput benefit from a total distributed registry in the MDS. Applications that fall between these classes benefit from a modified MOBA distributed registry.

We emphasize that a distributed Grid-based information service must be able to deal in most cases with organizational boundaries (Figure 5). All of the MDS-based solutions discussed above provide this nontrivial ability.

3.2 Grid-based Installation Service

In a Grid environment we foresee the following two possibilities for the installation of MOBA: (1) MOBA and Globus are already installed on the system, and hence we do not have to do anything; and (2) we have to identify a suitable machine on which MOBA can be installed. The following steps describe such an automatic installation process:

1. Retrieve a list of all machines that fulfill the installation requirements (e.g., Globus, JDK1.1, a particular OS-version, enough memory, accounts on which the user has access, platform-supported green-threads).
2. Select a subset of these machines on which to install MOBA.
3. Use a secure Grid-enabled ftp program to download MOBA in an appropriate installation space, and uncompress the distribution in this space.
4. Configure MOBA while using the provided auto-configure script, and complete the installation process.
5. Test the configuration, and, if successful, report and register the availability of MOBA on the machine.

3.3 Grid-based Startup Service

Once MOBA is installed on a compute resource and a user decides to run a MOBA place on it, it has to be started together with all the other MOBA services to enable a MOBA system. The following steps are performed in order to do so:

1. Obtain the authentication through the Globus Security service to access the appropriate compute resource.
2. List all the machines on which a user can start a MOBA place.
3. For each compute resource in the list, start MOBA through the Java CoG interface to the Globus remote job startup service.

Depending on the way the registry service is run, additional steps may be needed to start it or to register an already running registry within the MDS.

3.4 Authentication and Authorization Service

In contrast to the existing MOBA security system, the Grid-based security service is far more sophisticated and flexible. It is based on GSI and allows integration with public keys as well as with Kerberos. First, the user must authenticate to the system. Using this Grid-based single-sign on security service allows the user to gain access to all the resources in the Grid without logging onto the various machines on the Grid environment on which the user has accounts, with potential different user names and passwords. Once authenticated, the user can submit remote job request that are executed with the appropriate security authorization for the remote machine. In this way a user can access remote files, create threads in a MOBA place, and initiate the migration of threads between MOBA places.

3.5 Secure Communication Service

The secure communication can be enabled while using the GlobusIO library and sending messages from one Globus machine to another. This service allows one to send any serializable object or simple message (e.g., thread migration, class file transfer, and commands to the MOBA command interpreter) to other MOBA places executed under Globus-enabled machines.

4 Conclusion

We have designed and implemented migration system for Java threads as a plug-in to an existing JVM that supports asynchronous migration of execution context. As part of this paper we discussed various issues, such as whether objects reachable from the migrant should be moved, how the types of values in the stack can be identified, how compatibility with JIT compilers can be achieved, and how system resources tied to moving objects should be handled. As a result of this analysis, we are designing a JIT compiler that improves our current prototype. It will support asynchronous and heterogeneous migration with execution of native code. The initial step to such a system is already achieved because we have already implemented a distributed object system based on the JIT compiler to support selective migration. Although this is an achievement by itself, we have enhanced our vision to include the emerging Grid infrastructure. Based on the availability of mature services provided as part of the Grid infrastructure, we have modified our design to include significant changes in the system architecture. Additionally, we have identified services that can be used by other Grid application developers. We feel that the integration of a thread migration system in a Grid-based environment has helped us to shape future activities in the Grid community, as well as to make improvements in the thread migration system.

Acknowledgments

This work was supported by the Research for the Future (RFTF) program launched by Japan Society for the Promotion of Science (JSPS) and funded by the Japanese government. The work performed by Gregor von Laszewski work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Globus research and development is supported by DARPA, DOE, and NSF.

References

1. Anurag Acharya, M. Ranganathan, and Joel Saltz. Sumatra: A language for resource-aware mobile programs. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems*. Springer Verlag Lecture Notes in Computer Science, 1997.
2. Ole Agesen. GC points in a threaded environment. Technical Report SMLI TR-98-70, Sun Microsystems, Inc., December 1998. <http://www.sun.com/research/jtech/pubs/>.

3. Bozhidar Dimitrov and Vernon Rego. Arachne: A portable threads system supporting migrant threads on heterogeneous network farms. *IEEE Transaction on Parallel and Distributed Systems*, 9(5):459-469, May 1998.
4. M. Raşit Eskicioğlu. Design Issues of Process Migration Facilities in Distributed System. *IEEE Technical Committee on Operating Systems Newsletter*, 4(2):3-13, Winter 1989. Reprinted in *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press.
5. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann, 1998.
6. General Magic, Inc. Odyssey information. <http://www.genmagic.com/technology/odyssey.html>.
7. Satoshi Hirano. HORB: Distributed execution of Java programs. In *Proceedings of World Wide Computing and Its Applications*, March 1997.
8. Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-Grained Mobility in the Emerald System. *ACM Transaction on Computer Systems*, 6(1):109-133, February 1988.
9. David Kotz and Robert S. Gray. Mobile agents and the future of the internet. *ACM Operating Systems Review*, 33(3):7-13, August 1999.
10. Danny Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley Longman, Inc., 1998.
11. Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88-89, March 1999.
12. ObjectSpace, Inc. Voyager. <http://www.objectspace.com/products/Voyager/>.
13. M. Ranganathan, Anurag Acharya, Shamik Sharma, and Joel Saltz. Network-aware mobile programs. In *Proceedings of USENIX'97*, January 1997.
14. Tatsuro Sekiguchi, Hidehiko Masuhara, and Akinori Yonezawa. A simple extension of Java language for controllable transparent migration and its portable implementation. In *Springer Lecture Notes in Computer Science for International Conference on Coordination Models and Languages (Coordination99)*, 1999.
15. Tatsuro Sekiguchi. JavaGo manual, 1998. <http://web.yli.s.u-tokyo.ac.jp/amo/JavaGo/doc/>.
16. Kazuyuki SHUDO. shuJIT—JIT compiler for Sun JVM/x86, 1998. <http://www.shudo.net/jjit/>.
17. Kazuyuki Shudo and Yoichi Muraoka. Noncooperative Migration of Execution Context in Java Virtual Machines. In *Proc. of the First Annual Workshop on Java for High-Performance Computing (in conjunction with ACM ICS'99)*, Rhodes, Greece, June 1999.
18. Inc. Sun Microsystems. The Java HotSpot performance engine architecture. <http://www.javasoft.com/products/hotspot/whitepaper.html>.
19. Marvin M. Theimer and Barry Hayes. Heterogeneous Process Migration by Recompilation. In *Proc. IEEE 11th International Conference on Distributed Computing Systems*, pages 18-25, 1991. Reprinted in *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press.
20. Gregor von Laszewski and Ian Foster. Grid Infrastructure to Support Science Portals for Large Scale Instruments. In *Proc. of the Workshop Distributed Computing on the Web (DCW)*, pages 1-16, Rostock, June 1999. University of Rostock, Germany.
21. Gregor von Laszewski, Ian Foster, Jarek Gawor, Warren Smith, and Steve Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM 2000 Java Grande Conference*, San Francisco, California, June 3-4 2000. <http://www.extreme.indiana.edu/java00>.
22. James E. White. *Telescript Technology: The Foundation of the Electronic Marketplace*. General Magic, Inc., 1994.
23. Ann Wollrath, Roger Riggs, and Jim Waldo. A Distributed Object Model for the Java System. In *The Second Conference on Object-Oriented Technology and Systems (COOTS) Proceedings*, pages 219-231, 1996.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☒ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☒ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

THIS PAGE BLANK (USPTO)

XP-002309352

InfoGram: A Grid Service that Supports Both Information Queries and Job Execution

Gregor von Laszewski¹, Jarek Gawor¹, Carlos J. Peña^{1,2} and Ian Foster¹

¹ Argonne National Laboratory, 9700 S. Cass Ave, Argonne IL 60439, U.S.A.

² State University of New York at Stony Brook, Stony Brook, NY 11794, U.S.A.
gregor@mcs.anl.gov

Abstract

The research described in this paper is performed as part of the Globus Project. It introduces a new Grid service called InfoGram that combines the ability of serving as information service and as a job execution service. Previously, both services were architected and implemented within the Globus Toolkit as two different services with different wire protocols. The work demonstrates a significant simplification of the architecture while treating job submissions and information queries alike. The advantage of our service is that it provides backwards compatibility to existing Grid users while at the same time providing forwards compatibility to the emerging Web services world. Part of the work conducted within this effort is already reused by the current Open Grid Services Architecture prototype implementation.

1. Introduction

The Grid approach is an important development in the discipline of computer science and engineering [30]. It is making rapid progress on several levels, including the definition of terminology, the design of an architecture and framework [13], the application in scientific problems [5, 4], and the creation of physical instantiations of Grids on a production level [10, 3, 2]. Grids provide an infrastructure that allows for flexible, secure, coordinated resource sharing among dynamic collections of individuals, resources, and organizations.

Over the past few years, the Globus Project has developed the Globus Toolkit [18] that provides a basic Grid middleware toolkit, which includes elementary services to address Grid management issues related to resource management, security, information, and data management [30]. Two of the most important Grid services that are provided by the Globus Toolkit are the information service and the

job execution service.

The *information service* returns information about the capabilities and the state of the Grid infrastructure. The Globus Toolkit provides such an information service called Monitoring and Directory Service (MDS) [31, 7], formerly known as Metacomputing Directory Service.

The *job execution service* controls the submission and execution of jobs on remote machines. The Globus Toolkit provides such a service under the name Grid Resource Allocation and Management (GRAM) service [9]. GRAM processes requests for execution, performs resource allocation, monitors, and controls job execution. Furthermore, a limited amount of information related to the capabilities and availability regarding the job execution service for a Grid resource can be exposed through an information provider to the MDS. This information includes, for example, the name of the queue, details about the mode of operation, and other important features that may guide the process of job submission by the user. Authentication to MDS and GRAM are handled through the Grid Security Infrastructure (GSI) [6].

The information and job execution service have so far existed as separate services within the Globus Toolkit. Considerable software engineering effort is necessary to implement, maintain, and deploy these services while at the same time support interoperability. We argue that this complexity can be reduced significantly by alternative approaches to both protocol design and implementation. To test this hypothesis, we developed a prototype that promises a significant simplification in all aspects previously mentioned. We have termed our prototype InfoGram in order to acknowledge its dual purpose.

Our research has the following objectives and goals:

- Design of a simplified Grid service architecture to provide a unified service for information, monitoring, and job submission.
- Develop this service while providing backwards compatibility by adhering to standard Grid protocols.

Best Available Copy

- Support multiple information return request formats such as LDIF and XML.
- Improve the reliability of the job execution and in a second phase while replacing the protocol used to perform the Job submission with SOAP.
- Provide an open framework that can be easily adapted to interact with local schedulers and extract information through custom designed information providers.
- Provide a framework that is based on GSI and its application within the Globus toolkit to map Global Grid User Identifiers to local account names.
- Develop this service while providing forwards compatibility to Web services.
- Build a groundwork for a Web services based implementation of Globus services.

The rest of this paper is structured as follows: First, we discuss the Globus Toolkit services GRAM and MDS in more detail. We outline how the operational integration of these services is achieved in production Grids. Next, we present the enhancements to the GRAM service that allow constructing our InfoGram service. We demonstrate that our service provides a significant architectural simplification but at the same time provides enhancements currently not available in the Globus Toolkit. Additionally, we show that this new service can still be integrated into the existing MDS concept. Finally, we outline how such a service can be used as part of Grid applications.

2. Execution Service

To contrast our differences to the Globus GRAM it is necessary to revisit the architecture of the Globus GRAM service. The basic structure of a GRAM service (version 1.1.x) and its interaction with clients relevant for our discussion is depicted in Figure 1. A GRAM service provides the basic functionality for secure and uniform access to remote computational resources. The functionality of GRAM can be explained as part of a typical three tier architecture. Before we include our enhancements to this architecture (Section 6), we explain the functionality of each tier in more detail.

Client Tier. A client can submit a job to a remote resource and can check on its status either through polling the status of the job or through event notification to the client through the GRAM Service. To allow identification of the job, a job handle (often referred to GlobusID) is returned on job startup so that it can be used for later connection, including from other remote clients with appropriate authorization. For example, this job handle can be used to contact the job and issue a cancellation.

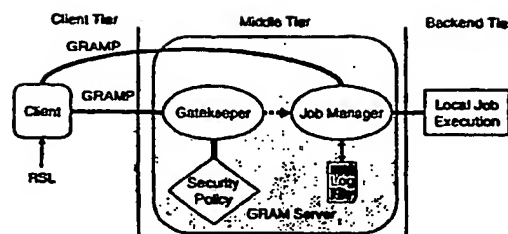


Figure 1. The Gram Architecture

Middle Tier. Internally, GRAM consists of a gatekeeper and a job manager. The gatekeeper is responsible for authentication with the client, performing a simple authorization based on mapping the authentication information into a local security context (e.g., a Unix login). After this initial security check, it starts up a job manager that interacts thereafter with the client based on the GRAM protocol (from now on referred to as GRAMP). Each job submitted by a client to the same GRAM will start its own job manager.

Backend Tier. Once the job manager is activated, it handles the communication between the client and the backend system on which the job is executed. The backend tier is easily portable to various scheduling systems. The Globus Toolkit services provide scheduling interfaces such as PBS, LSF, Condor, and Unix process fork [21, 18].

The GRAM service can be accessed with the help of a C or a Java application interface. This interface includes the ability to specify a job runnable on a particular resource with the help of a uniform Resource Specification Language (RSL). The RSL makes it possible to quickly and uniformly specify jobs to be run as part of a Globus enabled Grid. Simple tools are available to access the basic functionality also from the command line.

Although, we have in the past demonstrated mechanisms and protocols for application states and notification, such advanced functionality [32] has not yet been included in the Globus Toolkit.

3. Information Service

The basic structure of a Grid information service is defined in [31] and was further refined in [8]. A Grid information service requires:

- access to static and dynamic information regarding system components and services,

- a framework that fits well with the heterogeneous and dynamic nature of Grids, including decentralized maintenance and operation,
- scalability and performance,
- integration of a variety of information providers.

The Globus Project has developed a basic information service that addresses these requirements. The Globus Grid information service, MDS, contains two fundamental entities: distributed information providers and information aggregates. An information provider is a service that provides a subset of useful information about resources exploited by Grid users or Grid services. Examples of information that may be accessed through such an information provider is CPU, operating system, network, and file system information.

Additionally, the aggregate service is used to integrate a set of information providers that may be part of a virtual organization [14]. To increase the scalability of a distributed information service, the MDS provides an information caching function that allows viewing and querying the information about a resource from a cache. Furthermore, the newest implementation of a Grid information service that implements the framework proposed by the MDS concept integrates GSI to perform authentication.

The information contained within MDS can be queried and used to enable more sophisticated Grid services. More details about the protocols, the services, and the newest nomenclature can be found in [17, 7].

The research within this paper concentrates on the information provider itself, as we can create information aggregates through reuse of information providers to improve scalability. Furthermore, we argue that it is worthwhile to provide *google-like* services, as have been used in many previous Grid like projects [28, 29, 11].

4. Using GRAM and MDS in Production Grids

Figure 2 shows how the GRAM and the MDS services may be used in a simple production Grid. Our Grid consists of one virtual organization that maintains a number of compute resources. Each compute resource has the Globus GRAM and the Globus Resource Information Service (GRIS) that returns information related to the local resource installed.

In order for a client to perform a job execution and an information query, two different mechanisms for contacting these services must be used. Not only do the services operate through different ports, but they also use different protocols making the amount of code sharing for interpreting return values more complex. The installation of both services requires additional sophistication. We feel that the

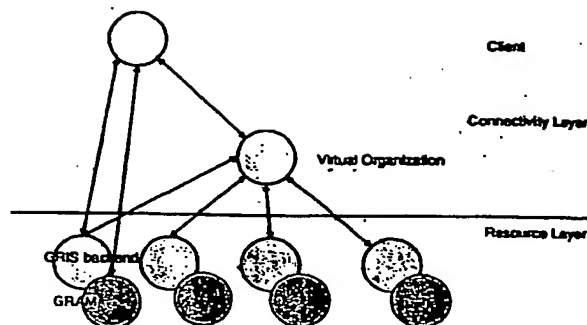


Figure 2. A sample interaction between a client, GRAM, and MDS

use of different technologies is in contrast with the desire to provide a minimal set of protocols and services for Grids as promoted by the Global Grid Forum and the Globus Project [14]. If we think abstractly about *job execution and an information service*, we must recognize that they are based on the same principle: A query formulated and submitted to a server followed by a stream of information that returns the result based on the query.

5 Addressing Requirements for the InfoGram Service

We have designed our InfoGram service according to a set of requirements determined by general software engineering practices which include factors such as quality, performance, reliability, security, and portability. All of these factors must be addressed within the realm of Grids. Nevertheless, we concentrate our efforts on the following issues.

5.1. Performance

An information and Job execution service must perform their tasks quickly. The elapsed time between job request and job submission must be as short as possible. At the same time, information within the system must be accessible quickly. For example, it may be inefficient to execute each time a user requests data the program creating the data or a query relayed to an external information service. A simple example will illustrate our point. Assume we have a large number of clients that need to know the CPU load of a remote compute resource. It would be wasteful to execute the command requesting the load every single time. Instead, it can be more efficient to cache this value within the information service, and only refresh this cache value

periodically. In order to prevent staleness of information we attach a time to live (TTL) value with the information. This value will tell us when a refresh of the information in the cache is necessary.

5.2. Quality of Information

Information within Grids may become quickly inaccurate. We often observe two cases. Case One: In the simplest form the information can be describe as binary system where the information is either accurate or inaccurate. Case Two: In many other situations the information may degrade over time in a discrete fashion. Thus, it is not unreasonable to attach a degradation function with the actual value of information that reflects the degree of degradation. This function may be influenced by time, system state, or prediction functions to derive a quality of information assessment. Often it is possible to attempt to derive such degradation information through observation or through mathematical models while performing self correction based on observation data. This is not unlike sophisticated data assimilation as used in climate research that corrects its values based on a comparison between observations and prediction models. The quality of information becomes important in case more sophisticated resource management strategies are developed. If I obtain an attribute such as "mean CPU load" from a Grid information service, it is beneficial to have the quality of the information attached. Knowing the standard deviation or knowing that the accuracy of the value is valid over the last hour or the last day is an important factor to create more sophisticated Grid services.

5.3. Security

Access to services such as the information and job execution needs to be performed securely. The Grid Security Infrastructure (GSI) provides us with an elementary framework for authentication. Nevertheless, authentication is only one problem to be addressed within Grids. In our framework, we strive to include authorization that allows us to specify contracts such as "allow access to this resource from 3 to 4 pm to user X." Additions to GSI and the use of more sophisticated authentication frameworks [27] may provide them in the future.

5.4. Portability

Protocol compatibility of these services is preserved with the Globus Toolkit while using the GRAM, and Grid Security Infrastructure (GSI) protocols. Future activities will include the integration of commodity protocols (such as SOAP) to provide interoperability to Web services and greater acceptance outside of the Grid community [15, 36, 38, 37].

5.5. Flexible and Extensible Information Model

One of the issues we face with information providers is the lack of a standard that is uniformly adhered by the community. We observe the use of CIM, MIB, MDS, or non standard or unorthodox display of information in tables. Although we believe that the creation of a consistent information model is an important one, we focus within this paper on the mechanism of delivering that information to the user. The reasoning for this strategy is that our InfoGRAM service provides the necessary mechanisms for delivering the information according to the information model used within the information provider. Our positive experience with the use of XML schemas as basis for the next generation of Information services makes us believe that it provides a viable alternative to the currently used LDAP schemas. Compatibility can be maintained while developing strict guidelines for the object definition by the Global Grid Forum.

Nevertheless, we believe that an additional requirement must be fulfilled to enhance the use and acceptance of Grids. We believe that the execution of untrusted applications in trusted environments is important to enable the use of Grids. We hope that through this feature the user community will increase dramatically based on software that is developed as part of our activities.

Providing such software will enable the creation of infrastructures that will promote Grids in new communities, which previously did not have the luxury to access high end resources. Besides making access to supercomputer centers for outside users much more feasible, we foresee that resource providers may be more willing to contribute resources otherwise not part of the national-scale Grids.

6. InfoGRAM Architecture

As pointed out earlier, we modified the architecture of the GRAM server and enhanced it substantially in order to fulfill the requirements described earlier. We added to the original architecture additional components, as shown by the shaded components in Figure 3, and describe these enhancements based on the functionality they are providing. These functionalities are centered on client interaction, logging and check pointing, job execution, information management, and configuration.

Logging and check pointing is enabled through a logging service. This service can receive logging events from several components. The log can either be stored in the middle tier, or on the backend tier. In either case the log can be used to restart our InfoGRAM service in case it needs to be restarted (e.g. the machine was shut down). In the same way it would be possible to use the logging service for check pointing of applications. Presently, we only record minimal information such as the command used and arguments

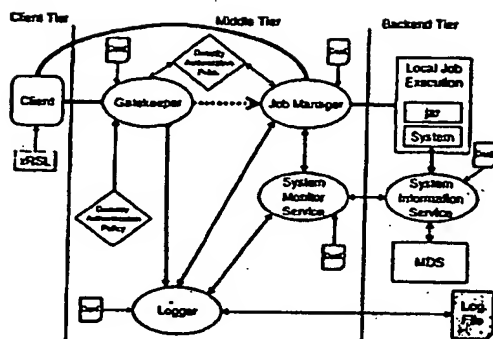


Figure 3. The InfoGram architecture that combines a GRAM service with an Information Service using only one protocol between client and server.

executed. We intend to use this logging service to provide simple Grid accounting.

6.1. Job execution

The execution of jobs is made more robust while integrating a logging and fault tolerance mechanism that allows to restart a job upon failure.

6.2. Information Service

As mentioned previously, the InfoGram Service contains several novel features in regards to the information service part it provides. These features are: An Information service that is integrated with GRAM providing backwards compatibility to MDS, and support of information caching and the retrieval of elementary information associated with the resource. Additionally, we are integrating in our service the feature of information degradation and self adaptation of information updates as discussed earlier.

Our information service is architected with two components (see Figure 3): the system monitor and the system information service. The monitor service controls initializing and caching the results requested by the clients. The system information service returns relevant information about the system resources, through either (a) calls to a system command via the Java runtime exec (b) a query to a function exposing Java runtime information such as load, memory, or disk space (c) or a read function from a file that is used by an information provider. A good example for an information provider is the Linux proc file system. As we have chosen an object oriented framework for our implementation, the integration of new information providers can be performed through the implementation of interfaces. This

will allow us to be able to provide a flexible and extensible information services framework.

```
class SystemInformation interface {
    String getKeyword();
    void setKeyword();
    Object queryState();
    Object updateState();
    Time ttl ();
    int validity();
    Public void setDelay(Time time);
    String setFormat(Format format);
    Time getAverageUpdateTime();
}
```

This interface allows us to generate new information providers in a fashion very similar to the current MDS model and its implementation. The method `queryState` is non blocking and returns valid information only when the information has been queried previously and the time to live (ttl) value has not expired. Otherwise, it throws an exception. Upon invocation of the `updateState` method, a blocking method is called that returns the appropriate information while also updating the time to live value. If multiple `updateState` methods are invoked, monitors are used to perform only one such update at a time. Additionally, we provide a delay that controls how many milliseconds must pass between consecutive calls of `updateState` before the actual information is obtained through a runtime exec call. This is useful in cases where users ask for information more frequently than it can be produced by the system.

6.3. Configuration

We provided a configuration component that allows us to setup the InfoGram service with ease. This component includes the possibility to configure the system monitor service with customized information providers similar to the MDS. This configuration file contains the following parameters:

TTL: the lifetime in millisecond of each data generated by the specific key word; 0 specifies execution of the key-word every time it is requested.

Keyword: the keyword that will be used in an RSL string to identify the mapping to a real program or a Java application to be executed in the background.

Executable Path: the full executable path and name with arguments, machine dependant that is associated with the keyword.

Best Available Copy

Table 1. The InfoGram configuration file provides a mapping between keywords and information providers

TTL	Keyword	Command
60	Date	date -u
80	Memory	/sbin/sysinfo.exe -mem
100	CPU	/sbin/sysinfo.exe -cpu
0	CPUload	/usr/local/bin/cpuload.exe
1000	list	/bin/ls /home/gregor

We provide an example for the information represented within such a configuration file in Table 1. As the keyword identifies the information obtained with the program we will refer to it from now on as a key information provider. Each attribute within a key information provider is augmented with a namespace conform to the keyword. Thus the attribute "total" in the "Memory information provider" would be referred to as `Memory:total`.

6.4. Caching and Information Degradation

The caching functionality is similar to that of the MDS 2.0. Nevertheless, queries to the information service are simple all-or-nothing queries based on the keywords used within the configuration files. That means, all attributes that are obtained through the command associated with a keyword will be returned. Based on this simple model, the caching of information is easily possible. Additionally, we have the option to augment each attribute that is returned within a key information provider with a degradation function or a quality of information value. Selecting similar information attributes can then be performed on the quality of the information provided.

6.5. Service Reflection

Each information service can be queried and a client may inspect the schema that is returned by the information service. Thus it will allow developers to design programs that can be flexible to the actually used information schema. We believe that reflection and introspection of the capabilities of an execution and information service will become increasingly important with the increased number of available Grid services.

6.6. Client Interaction through xRSL

Although we developed our first prototype architecture as a Web service, we believed at the time that it would provide to big of a departure from the existing Globus Toolkit.

We thought that most important for the acceptance of our information service, is the recognition that the Globus Toolkit reached ubiquity within the community, and that the Globus protocols should be reused.

Thus, as we wanted to maintain a degree of backwards compatibility, we decided not to chose a pure Web services-based implementation that uses only WSDL [36], XML-schema [39], and XML query. We felt that such an effort could be performed in a second step (as it is now performed as part of the Open Grid Service Architecture [1]). Instead of using URIs to formulate job submission and information queries, we argued that users of the Globus Toolkit are sufficiently familiar with RSL. Therefore, it was most natural to extend RSL with the more advanced features we have introduced so far. We added the following tags to the Globus RSL: *schema*, *info*, *filter*, *response*, *performance*, *quality*, and *format*. We call the result xRSL.

Info. The *info* tag is followed by the *key* as specified in the configuration file, defining a mapping between the keyword and the command to be executed. If it is set to (*info=all*), all commands are executed. Commands can be selectively queried while concatenating multiple *info* tag queries, for example, (*info=Memory*) (*info=CPU*). A special value for the *info* tag is (*info=schema*). This returns a hierarchical schema that contains all objects associated with the keywords and lists properties of their attributes.

Response. The *response* tag defines the behavior with respect to the information caching. Thus, with (*response=immediate*) the commands associated with the *info* tag are executed immediately regardless of the time to live. This will also update the cached values. Using (*response=cached*) will return the information from the cache value if it is valid; otherwise it will update the cache first. Using (*response=last*) will return the value stored last in the cache without updating it.

Quality. The *quality* threshold tag provides the possibility to specify a percentage number that gives additional guidance if a cached value should be returned or if the information needs to be refreshed before return. Currently, we define the following semantic. If the degradation function of any of its returned attributes is below that threshold, this attribute is regenerated by the associated command.

Performance. The *performance* tag returns the number of seconds and the standard deviation about how long it takes to obtain a particular information value. The performance of a command and its attributed values is measured and catalogued during runtime.

Format. The *format* tag defines the format in which the information is returned. The supported formats are LDIF and XML. Nevertheless, it is straightforward to support other formats such as DSML.

Extensions. We are planning to extend our ex-

isting timeout tag with an additional action tag upon reaching this timeout. For example, the RSL (executable=command) (timeout=1000) (action=cancel) would cancel the command specified through the RSL, while (action=exception) would throw an exception if the command has not completed its execution, but the execution of the command itself would be continuing.

Advantages. The advantages of this information stem from the simplification of the architecture bound to the delivery of an integrated job submission and information service. Querying the information is handled by clients much as the execution of jobs. Moreover, this information service can easily be integrated into the Globus MDS information service architecture.

In summary, we have explored changes to MDS at the protocol and the implementation level. At the protocol level we have replaced an LDAP search query with a "query" cast as a simple job submission through RSL. This new query mechanism is based on At the implementation level, we have replaced the modular, configurable MDS information provider architecture with a less complex, even more modular, configurable architecture that we believe fulfills, in a straightforward fashion, the Grid designers quest for an easy to use and maintain information service. As part of this implementation effort we have also explored more advanced features for dealing with caching of the information based on quality augmentations to the data itself. The result of our simplified architecture is presented in Figure 4 and contrasts our earlier Figure 2. We believe that although the number of the components within our Info Gram service increased the overall complexity of the combined service is lower than the current provided solution.

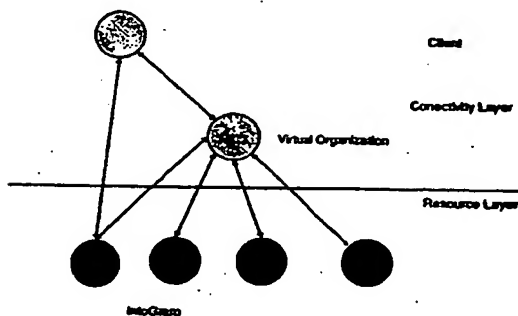


Figure 4. The new InfoGram service reduces the number of protocols and components in a Grid.

7. Implementation

Although our services can be implemented in any other language we have chosen to prototype them in Java. It is a straightforward engineering exercise to implement them in C.

The Java platform enhances the functionality of our service based on the use of additional features that are otherwise not available in C. Thus, we were able to achieve:

- Delivery of a pure Java Information and GRAM service providing cross-platform portability, which includes the Windows Operating System.
- Delivery of a Web-enabled installation service that can deploy the InfoGram service with low overhead on installation time and administrative burden.
- Execution of untrusted applications in trusted environments on remote machines as part of the Java Virtual Machine model.

To support the development of the previously outlined service, we have performed significant enhancements to the Java CoG Kit that is maintained as part of the Globus Project. These enhancements are focusing on the job submission, deployment, logging, security, and information service. Whenever possible, we use standard Java packages to reduce the amount of codebase that must be maintained by us. This includes logging [20, 25] and security [19, 26].

In a first step, we have implemented a pure Java implementation a Globus GRAM [16, 9] service that provides much the same functionality than its C-based counterpart. In order to support interoperability and compatibility, we based the design directly on the architecture of the C GRAM service. It contains a gatekeeper, job manager, and a local job execution process. We name this service J-GRAM.

Job Submission.

This Job Execution service within J-GRAM is protocol-compatible with the "C-GRAM" distributed with the Globus Toolkit. At present, we investigate the implementation of major GRAM functionality, such as the support for gridmaps, which map user certificates to local user IDs, as well as the possibility to interfaces easily to schedulers. We learned from this prototype that it is possible to provide a service in Java that mimics the behavior of C-GRAM.

Besides the invocation of executables from precompiled native code, our J-GRAM service enhances the normal Globus GRAM service by being able to execute pure Java code submitted as Java jar files. To enable the execution of jar files as part of the J-GRAM service, a variety of changes were necessary. We extended the functionality of the job manager to start up the code embedded in

a jar file that was submitted through an RSL call such as, (executable=myJavaApplication.jar)

In order to run Java applications, one method is to execute the code in the same JVM as the rest of the components are running. An alternative is to separate the execution of the job into a JVM to increase security [19, 24]. We provide the ability to configure the job manager to run in either of these modes. The Grid administrator must decide which mode should be run. The execution of system commands is performed through the runtime.exec() call. It is possible to redirect I/O to and from the client. The functionality is equivalent to the one from the C GRAM service with exception that DUROC is not supported. As the Globus Project will replace it in the near future, we have decided to refer to full delegation to a C Globus GRAM in order to provide this functionality. Therefore, it is still possible to start up MPIC11-G2 jobs [22].

Deployment. We have demonstrated this service at SC2001 and featured the ease of installation of such a service while using the Java framework deployment methods known as Web Start. Using this advanced deployment protocol, we are also able to maintain the upgradeability with more ease and to provide future solutions for automatically upgrading such services in production Grids. This feature is naturally supported while choosing Java as an implementation and deployment platform. Such sophisticated approaches require much more effort in traditional operating systems.

Logging. We are in the process of refining a logging mechanism for the execution of jobs assists in the fault recovery abilities of GRAM, as well as the possibility of logging authenticated information queries to guide the use as part of intelligent scheduling services.

Secure Sandboxing. In traditional programming languages, such as C, C++, and FORTRAN, it is difficult to execute untrusted applications in a trusted environment similar to the one the Grid provides. With a JVM, however, we are able to enable a trust relation between an untrusted client application to be executed in a trusted environment. Additionally, we were able to package a gatekeeper with non-root access rights in a jar file that can be easily installed in one environment. J-GRAM can be configured in various ways. We can either execute each job in the already running JVM or start up a number of external JVM to execute a jar file in an even more restrictive environment.

Portability. Other advantages (that are based on the use of Java) are the immediate availability of an information service on the Windows operating system. Other benefits are introduced by providing authorization mechanisms as part of this service, which can be supported by the Java platform.

InfoGram. In a second step we have prototyped much of the functionality described within this paper to enable

the InfoGram service. We have obtained good experience to return information queries in LDIF and XML.

8. Application

Currently, the J-GRAM service has already been used in several projects, one of which is the emerging OGSA framework [12] that has been developed after our investigations.

We have tested our InfoGram prototype on an application that we have termed a sporadic Grid. Such a Grid is created just for a short period of time during sophisticated experiments at synchrotrons or photon sources [35, 34]. To implement such a service we need a simple architecture that contains a set of advanced Grid services that are useful for supporting the creation and maintenance of sporadic Grids. Our InfoGram service provides such a service. As we are able to distribute it as a pure Java application, it will be easy to install it on a number of machines or access it through Web-browsers.

We will extend our efforts to support computationally mediated sciences [40]. In this technique, a focused electron probe is sequentially scanned across a two dimensional field of view a thin specimen, and at each point on the specimen a two dimensional electron diffraction pattern is acquired and stored. The analysis of the spatial variation in the electron diffraction pattern allows a researcher to study the subtle changes resulting from microstructural differences, such as ferro and electro magnetic domain formation and motion at unprecedented spatial scales. We will provide the computational Grid infrastructure for these classes of experiments.

9. Related Work

Parallel to the research described in this paper, modifications to GRAM1.0 were performed by colleagues within the Globus Project together with the Condor team at the University of Wisconsin. This modified version of GRAM is available as part of the Globus 2.0 release. We are protocol compatible to that version. Most recently, the Globus Project, started together with IBM on the Open Grid Services Architecture. Our work was performed before OGSA. Lessons learned from our activities should have influence on the OGSA work. The current OGSA prototype implementation uses the J-GRAM service, as well as the GSI security provided through the Java CoG Kit [33].

10. Status and Future Plans

The work performed within this research activity explored new concepts that we expect to be considered in future Globus Toolkit developments. Future research activities will include exploration of conceptual issues identified

within this paper, as well as their implementation as part of prototype and Globus toolkit developments. On the conceptual level, we will investigate the explicit guidelines for system designers to choose the right configuration for setting up the InfoGram Service with the appropriate parameters and configuration files. We will perform further simplifications on the J-GRAM architecture while using only one port to communicate between job managers and clients. For compatibility reasons, we have not yet been able to perform this change. Improved fault tolerance will allow for automatic restart capabilities enabled through checkpointing. We are improving our code and hope to integrate it in either the Globus Toolkit or the OGSA framework. Several features, such as the use of the performance tag and the information degradation, are integrated at the moment. We are also experimenting with integration of our framework in Web services and JXTA [23].

11. Discussion and Conclusion

We feel that we have contributed to several areas within Grid computing. First, we identified that it is possible to design an Information system and a Job submission service that simplifies the architecture of the services provided by the Globus Toolkit. Through the extension of the RSL it will be easy for current Globus Toolkit users to adapt their code to use this information query. Second, we provide the possibility of being protocol compatible to the Globus Toolkit, while being able to integrate our information provider in the existent MDS. Therefore, we provide the option to move to a different Information provider while enabling a gradual transition. New information providers could be integrated easily in this information service framework. Third, we already integrated in the current Java CoG Kit our J-GRAM service that allows executing untrusted applications in trusted environments. This service is naturally able to run on Windows platforms and can be used to support sporadic Grids as defined in the paper. Forth, we set the stage for a multi protocol support for Grid information services that may export their data in LDIF or XML-schema.

We presented suggestions for enhancing the Globus Toolkit and believe that future development on Globus GRAM can benefit from our research on sporadic Grids. We believe that the Open Grid Services Architecture will benefit from this work performed over the last year. In particular the simplified InfoGram service can be used as an elementary replacement for a lightweight job execution and information service. It is straight forward to cast the InfoGram in WSDL. Considerable software engineering effort is necessary to implement, maintain, and deploy these services while at the same time support interoperability.

Acknowledgments

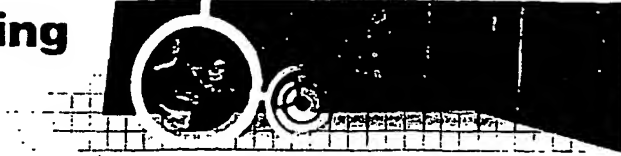
This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Project research and development. The work of Carlos J. Peña was partially funded by the DOE ERULF program. We thank Trilok Velingetti for his help during the implementation. We thank Ian Foster, Dennis Gannon, Peter Lane, Nell Rehn, Mike Russell for the valuable discussions during the course of the ongoing development. This work would not have been possible without the help of the Globus team. Globus Toolkit and Globus Project are trademarks held by the University of Chicago.

References

- [1] The physiology of the grid: An open grid services architecture for distributed systems integration. Available from <http://www.globus.org/research/papers/ogsa.pdf>.
- [2] EUROGRID: Application Testbed for European Grid Computing, 2001. <http://www.eurogrid.org/>.
- [3] Information Power Grid Engineering and Research Site, 2001. <http://www.ipg.nasa.gov/>.
- [4] Neesgrid homepage. <http://www.neesgrid.org/>, March 2002.
- [5] G. Allen, W. Benger, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf. The cactus code: A problem solving environment for the grid. In *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on*, pages 253–260, San Francisco, August 2000.
- [6] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. Design and deployment of a national-scale authentication infrastructure. *IEEE Computer*, 33(12):60–66, 2000.
- [7] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *10th IEEE International Symposium on High Performance Distributed Computing*, pages 181–184, San Francisco, CA, August 7–9 2001. IEEE Press. www.globus.org.
- [8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. pages 181–184, 2001.
- [9] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, Lecture Notes on Computer Science, 1998.
- [10] Doe science grid. <http://www.doesciencegrid.org/>.
- [11] G. Fagg, K. Moore, and J. Dongarra. Scalable networked information processing environment (snipe). *International Journal on Future Generation Computer Systems*, 15:595–605, 1999.

- [12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, The Globus Project, Jan. 2002. <http://www.globus.org/ogsa>.
- [13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.
- [14] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200-222, 2001. www.globus.org/research/papers/anatomy.pdf.
- [15] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvez. Programming the grid: Distributed software components, p2p and grid web services for scientific applications. <http://www.extreme.indiana.edu/an/papers/ProgGrids.pdf>.
- [16] The Globus GRAM Web Page. www.globus.org/gram.
- [17] The Globus MDS Users' Guide. www.globus.org/mds/mdsusersguide.pdf.
- [18] The Globus Project Web Page. www.globus.org.
- [19] IAIK Java Cryptology, 2001. <http://jcewww.iaik.at/>.
- [20] Logging Toolkit for Java, July 2001. <http://www.alphaworks.ibm.com/tech/loggingtoolkit4j>.
- [21] J. Kaplan and M. Nelson. A comparison of queueing, cluster and distributed computing systems, 1994.
- [22] N. Karonis. MPICH-G2 Web Page, 2001. <http://www.hplab.nyu.edu/mpi/>.
- [23] N. Krishnan. The jxta solution to p2p. October 10 2001.
- [24] P. Lipp. *Sicherheit und Kryptographie in Java. Einführung, Anwendung und Lösungen*. Addison-Wesley, 2000.
- [25] LogKit Developer Documentation, 2001. <http://jakarta.apache.org/avalon/logkit/whitepaper.html>.
- [26] S. Oaks and N. Inc. *Java Security*. Java series. O'Reilly, Cambridge, 1998. <http://www.netlibrary.com/>.
- [27] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based Access Control for Widely Distributed Resources. In *Proc. 8th Usenix Security Symposium*, Washington, DC, August 23-26 1999. <http://www-itsg.lbl.gov/Akenti/papers.html>.
- [28] G. von Laszewski. *A Parallel Data Assimilation System and its Implications on a Metacomputing Environment*. PhD thesis, Syracuse University, Nov. 1996.
- [29] G. von Laszewski. A loosely coupled metacomputer: Co-operating job submissions across multiple supercomputing sites. *Concurrency: Experience, and Practice*, 11(15):933-948, 1999. www.cogkits.org.
- [30] G. von Laszewski. Grid Computing: Enabling a Vision for Collaborative Research. In *Conference on Applied Parallel Computing, 3rd CSC Scientific Meeting, Lecture Notes*, Espoo, Finland, 15 - 18 June 2002. Springer.
- [31] G. von Laszewski, S. Fitzgerald, I. Foster, C. Kesselman, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proc. 6th International Symposium on High-Performance Distributed Computing*, pages 365-375, Portland, OR, August 5-8 1997. IEEE. [ftp://ftp.globus.org/pub/globus/papers/hpdc97-mds.pdf](http://ftp.globus.org/pub/globus/papers/hpdc97-mds.pdf).
- [32] G. von Laszewski and I. Foster. Grid infrastructure to support science portals for large scale instruments. In *Proc. of the Workshop Distributed Computing on the Web*. University of Rostock, Germany, June 21-23 1999. <http://www.mcs.anl.gov/laszewski/papers/rostock.pdf>.
- [33] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643-662, 2001. <http://www.globus.org/cog/documentation/papers/cog-cpe-final.pdf>.
- [34] G. von Laszewski, I. Foster, J. A. Insley, J. Bresnahan, C. Kesselman, M. Su, M. Thiebaut, M. L. Rivers, I. McNulty, B. Tieman, and S. Wang. Real-time analysis, visualization, and steering of microtomography experiments at photon sources. In *SIAM99*. SIAM, 1999.
- [35] G. von Laszewski, M. L. Westbrook, C. Barnes, I. T. Foster, and E. M. Westbrook. Using computational Grid capabilities to enhance the capability of an X-ray source for structural biology. *Cluster Computing*, 3(3):187-199, 2000.
- [36] Web services description language (wsdl) 1.1. <http://www.w3.org/TR/wsdl>. W3C.
- [37] www-4.ibm.com/software/solutions/webservices/.
- [38] Web services inspection language (wsil). <http://xml.coverpages.org/IBM-WS-Inspection-Overview.pdf>.
- [39] XML Schema, Primer 0, 1, and 3, 2001. <http://www.w3.org/XML/Schema>.
- [40] N. J. Zaluzec and G. von Laszewski. Computationally mediated sciences. in preparation, September 2002.

About Grid computing



What would it mean if you could:

- Analyze the value of an investment portfolio in minutes rather than hours?
- Unite research teams with others around the world to take advantage of the most up-to-date learnings?
- Significantly accelerate the drug discovery process?
- Scale your business to meet cyclical demand?
- Cut the design time of your products in half while reducing the instances of defects?

Government labs and scientific organizations have been using grid technologies for several years, solving some of the most complex and important problems facing mankind. Now grid computing is becoming a critical component of day-to-day business. Today's challenging business climate requires continuous innovation to differentiate products and services. Businesses must adjust dynamically and efficiently to marketplace shifts and customer demands.

IBM's response to these customer needs is what e-business on demand is all about. There's a profound shift afoot in how computing is used — even in basic assumptions about how it's accessed and paid for. Grid computing can bring tremendous productivity and efficiency to organizations facing the challenges of an on demand world.

IBM has practical information on grid computing

Find out what a grid is.

→ [What is grid computing?](#)

Learn how IBM uses grid.

→ [IBM and grid](#)

Learn about the significant productivity and efficiency gains that grid can offer businesses today.

→ [Grid benefits](#)

Get answers to frequently asked questions for businesses just considering grid computing, as well as those taking the next steps in unleashing grid power.

→ [Frequently asked questions](#)

BEST AVAILABLE COPY

An Overview of an Architecture Proposal for a High Energy Physics Grid

A. Wäänänen¹, M. Ellert², A. Konstantinov³, B. Kónya⁴, and O. Smirnova⁴

¹ Niels Bohr Institutet for Astronomi, Fysik og Geofysik,
Blegdamsvej 17, DK-2100, Copenhagen Ø, Denmark

² Department of Radiation Sciences, Uppsala University,
Box 535, 751 21 Uppsala, Sweden

³ University of Oslo, Department of Physics,
P. O. Box 1048, Blindern, 0316 Oslo, Norway

⁴ Particle Physics, Institute of Physics, Lund University,
Box 118, 22100 Lund, Sweden

Abstract. This document gives an overview of a Grid testbed architecture proposal for the NorduGrid project [1]. The aim of the project is to establish an inter-Nordic¹ testbed facility for implementation of wide area computing and data handling. The architecture is supposed to define a Grid system suitable for solving data intensive problems at the Large Hadron Collider at CERN [2]. We present the various architecture components needed for such a system. After that we go on to give a description of the dynamics by showing the task flow.

1 Introduction

This document assumes basic knowledge of the computing *Grid* concept, which is a paradigm for the modern distributed computing and data handling. For a general introduction to Grid computing the reader is referred to eg. [3]. The most common starting point for constructing a computing Grid is the Globus Toolkit² [4]. This toolkit provides a Grid API and developing libraries as well as basic Grid service implementations.

The NorduGrid project is a common effort by the Nordic countries to create a Grid infrastructure, making use of the available middleware. Through the European DataGrid project (EDG) [5] the NorduGrid project has had extensive experience with the Globus Toolkit and with deploying and using a Grid Testbed. During this we have found some shortcomings in the Globus Toolkit and some problems with the EDG Testbed architecture that we would like to address on a Grid testbed in the Nordic countries. In this paper we present a proposal for a Grid architecture for a production testbed at the LHC experiments. It is not the intent to define a general Grid system, but rather a system specific for batch processing suitable for problems encountered in High Energy Physics. Interactive

¹ The term *Nordic* covers the countries: Denmark, Norway, Sweden and Finland.

² Globus Project and Globus Toolkit are trademarks held by the University of Chicago.

and is usually realized by a NFS server. A remote SE is usually a stand-alone machine running eg. GridFTP [6] server with local file storage. Data replication is done by services running on the SE.

A dedicated pluggable GridFTP server has been developed for use on the SE. At the moment a simple file access plugin exists. The main reason for this is to have a way to provide a consistent certificate-based data access to the data. At least one other Grid solution to a certificate-based filesystem exists [7]. One advantage of the GridFTP approach is that it is done entirely in user space and thus is very portable.

2.3 Replica Catalog – RC

The information about replicated data is contained in the *Replica Catalog* (RC). This is an entirely add-on component to the system and as such is not a requirement.

2.4 Information System – IS

A stable, robust, scalable and reliable information system is the cornerstone of any kind of Grid system. Without a properly working information system it is not possible to construct a functional Grid. The Globus Project has laid down the foundation of a Grid information system with their LDAP-based Metacomputing Directory Service (MDS) [9]. The NorduGrid information system is built upon the Globus MDS.

The information system described below forms an integral part of the NorduGrid Testbed Architecture. In our Testbed, the NorduGrid MDS plays a central role: all the information related tasks, like resource-discovery, Grid-monitoring, authorized user information, job status monitoring, are exclusively implemented on top of the MDS. This has the advantage that all the Grid information is provided through a uniform interface in an inherently scalable and distributed way due to the Globus MDS. Moreover, it is sufficient to run a single MDS service per resource in order to build the entire system. In the NorduGrid Testbed a resource does not need to run dozens of different (often centralized) services speaking different protocols: the NorduGrid Information System is purely Globus MDS built using only the LDAP protocol.

The design of a Grid information system is always deals with questions like how to represent the Grid resources (or services), what kind of information should be there, what is the best structure of presenting this information to the Grid users and to the Grid agents (i.e. Brokers). These questions have their technical answers in the so-called LDAP schema files. The Globus Project provides an information model together with the Globus MDS. We found their model unsuitable for representing computing clusters, since the Globus schema is rather single machine oriented. The EDG suggested a different CE model which we have evaluated [8]. The EDG's CE-based schema fits better for computing clusters. However, its practical usability was found to be questionable due to improper implementation.

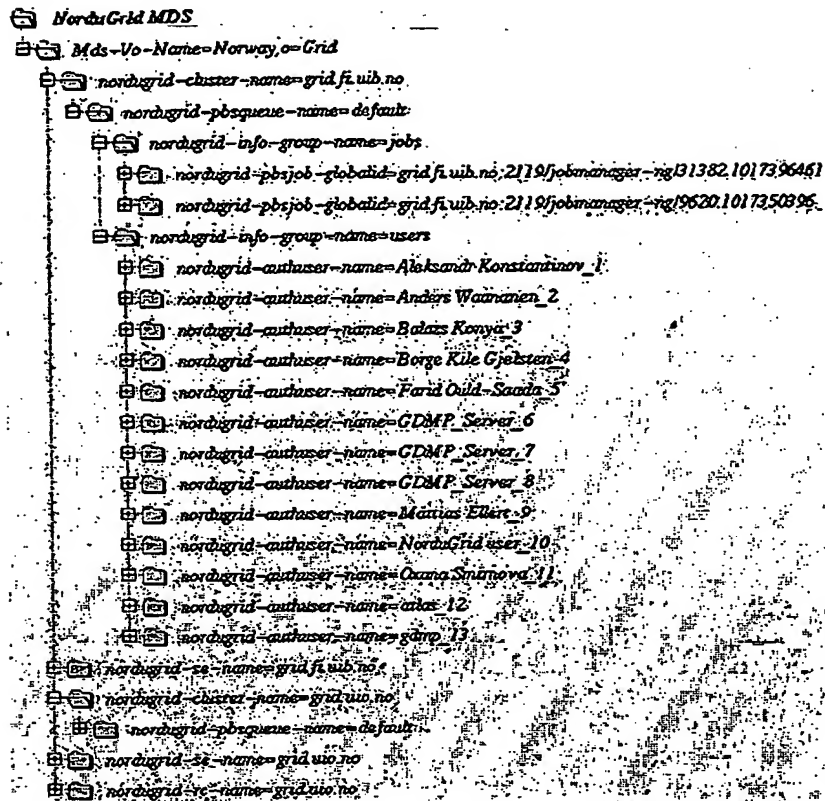


Fig. 1. The Norway branch of the NorduGrid MDS tree

2.5 Grid Manager – GM

In our model, job management is handled by a single entity which we call the *Grid Manager* (GM). It is the job of the GM to process user requests and prepare them for execution on the CE. It also takes care of post-processing the jobs before they leave the CE. In the Globus Toolkit context, the GM takes care of what is normally done by the Globus job-manager. In fact it is installed in a similar way to a standard Globus jobmanager and can work perfectly together with already existing jobmanagers. Authorization and authentication is still done by the Globus gatekeeper.

The status of each job is recorded in a special status directory which also contains control files needed by the GM.

```

Distinguished Name = [nordugrid-pbsjob-globalid=grid.uio.no:2119/jobmanager-ng/31633.1017417271]
objectClass = [pbs]
objectClass = [nordugrid-pbsjob]
nordugrid-pbsjob-globalid = [grid.uio.no:2119/jobmanager-ng/31633.1017417271]
nordugrid-pbsjob-globalowner = [/o=grid/c=nordugrid/ou=quark.ln.se/cn=Salaks Kõrva]
nordugrid-pbsjob-jobname = [sleep job]
nordugrid-pbsjob-submissiontime = [20020323165433Z]
nordugrid-pbsjob-execuser = [default]
nordugrid-pbsjob-exechost = [grid.uio.no]
nordugrid-pbsjob-csname = [MILANS_2]
nordugrid-pbsjob-startperiod = [00:00:00]
nordugrid-pbsjob-expirationtime = [00:05:42]
nordugrid-pbsjob-machname = [pbs]
nordugrid-pbsjob-comment = [job started on Fri Mar 23 at 16:54]
nordugrid-pbsjob-status = [Not yet]
nordugrid-pbsjob-error = [Not yet]
nordugrid-pbsjob-submissionid = [Not Yet Implemented]
Mdn-suffixes = [20020323160226Z]
Mdn-suffixes = [20020323160256Z]

```

Fig. 3. Example nordugrid-pbsjob entry

all job requests as well as data payload had to pass through, would be a single point of failure and non-scalable.

The NorduGrid UI is at present command-line driven, while a web based solution is foreseen in the future. The UI is responsible for generating the user request in a *Resource Specification Language* (RSL) based on the user input. The RSL we use has additional attributes to those provided by the Globus Toolkit [12]. This xRSL has been enhanced to support enriched input/output capabilities and more specification of PBS requirements. All unneeded Globus attributes has been deprecated.

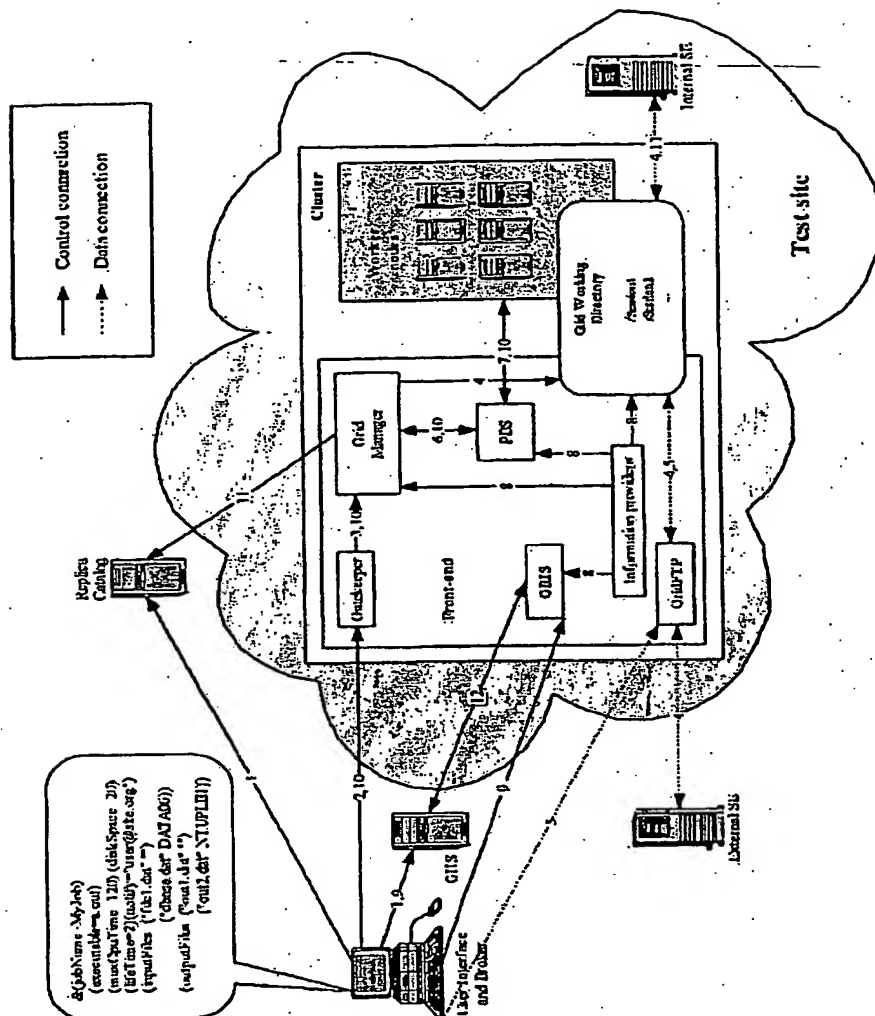


Fig. 4. NorduGrid task flow

10. User Interface may cancel jobs by sending cancellation commands through the Gatekeeper to the Grid Manager. The Grid Manager will then take care of the job clean up

5. The European DataGrid: <http://www.eu-datagrid.org/>
6. W. Allcock, J. Bester, J. Breshnahan, A. Chervenak, L. Liming, S. Tuecke: GridFTP: Protocol Extensions to FTP for the Grid
<http://www.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf>
7. A. McNabb: *SlashGrid - a framework for Grid aware filesystems*.
<http://www.gridpp.ac.uk/slashgrid/>
<http://www.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf>
8. B. Kónya: *Comments on the Computing Element Information Provider of the EU DataGrid Testbed 1*.
http://www.nordugrid.org/documents/comments_on_ceinfo.pdf
9. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman: *Grid Information Services for Distributed Resource Sharing*. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
10. I. Foster, C. Kesselman, S. Tuecke.: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International J. Supercomputer Applications, 15(3), 2001.
11. T. Miao: *LDAPExplorer*. <http://igloo.its.unimelb.edu.au/LDAPExplorer/>
12. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke: *A Resource Management Architecture for Metacomputing Systems*. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62-82, 1998.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☒ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☒ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

STAGED SIMULATION FOR IMPROVING SCALE AND PERFORMANCE OF WIRELESS NETWORK SIMULATIONS

Kevin Walsh
Ermin Gün Sirer

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501, U.S.A.

ABSTRACT

This paper describes *staged simulation*, a technique for improving the run time performance and scale of discrete event simulators. Typical wireless network simulations are limited in speed and scale due to redundant computations, both within a single simulation run and between successive runs. Staged simulation proposes to reduce the amount of redundant computation within a simulation by restructuring discrete event simulators to operate in stages that precompute, cache, and reuse partial results. This paper presents a general and flexible framework for staging, and identifies the advantages and trade-offs of its application to wireless network simulations. Experience with applying staged simulation to the ns2 simulator shows that it can improve execution time by an order of magnitude in typical scenarios and make feasible the simulation of large scale wireless networks.

1 INTRODUCTION

The design and evaluation of distributed systems and network protocols relies to a large extent on network simulation. Traditional network simulators, however, do not run efficiently or scale well with increasing simulation size.

A significant source of inefficiency in discrete event simulators is redundant computation. We identify two different classes of redundancy in traditional discrete-event simulators. The first class of redundant computation occurs within a single run of the simulator. Traditional network simulators reevaluate complex functions whenever their results may have changed, even though in reality the results may have changed very little, if at all, since the last time they were evaluated. A second class of redundant computation stems from a lack of retained information between multiple runs of the simulator. Executing each simulation independently and without the benefit of past runs leads to computing many functions from scratch in each run. These two sources of redundancy pose significant bottlenecks for

wireless network simulations, where network parameters change frequently.

This paper introduces *staged simulation*, a general technique to improve the scale and performance of wireless network simulation by exposing, identifying, and eliminating sources of redundant computation. Staging involves restructuring the events in a discrete-event simulator into an equivalent set of sub-computations, caching their results, and reusing them whenever matches are identified. We introduce three techniques, called *function decomposition*, *refinement*, and *batching* to complement function caching and improve its effectiveness. We apply these techniques both within a single simulation, a technique called *intra-simulation staging*, and between multiple similar runs of the simulator, called *inter-simulation staging*.

We have applied staging to the event processing engine of ns2 (VINT 1995), a well-established simulator whose design is typical of many discrete event simulators. Staging improved execution time by an order of magnitude over the standard ns2 implementation under typical simulation scenarios. As a natural consequence of eliminating redundant computation, staging in ns2 also reduced the running time from $O(n^2)$ in the size of the simulated wireless network to $O(n)$, making feasible large scale simulations with tens of thousands of nodes. Staging maintains strict compatibility with existing simulation scripts and extensions, with no loss in simulator generality or accuracy. More advanced and specialized simulation engines can benefit equally from staging. Specifically, we expect to see a comparable speedup and improvement in scalability in parallel and distributed wireless network simulators.

The contributions of this paper are as follows. First, we identify and expose a general technique for improving discrete event simulator performance. Second, we show how common simulation scenarios can benefit substantially from our optimization techniques. These benefits include drastically reduced simulator run time and good scalability without changing the simulator interface or degrading result accuracy. Finally, we validate our technique through system-

atic application to wireless simulation in a well-established network simulator.

2 THE STAGING APPROACH

The goal of staging is to eliminate redundant or nearly redundant computations in simulations. Traditional wireless simulators perform many redundant computations within a single run. Examples of common redundancies include sending packets along a particular path or computing neighbor sets. Similarly, across multiple runs of a simulator we find a large overlap in computation, especially when numerous runs of a simulation are made with only slightly varying parameters. For example, studies of proposed ad hoc routing protocols typically call for several sets of simulation runs, each set evaluating the effect of a single protocol or topology parameter (see, for example, Broch et al. 1998 and Royer and Toh 1999). In all, many dozens or hundreds of runs might be executed with very similar input parameters.

The simplest, most fundamental technique for eliminating redundant computations is function caching. This space-for-time trade off involves caching the results of idempotent functions and later reusing those results whenever the same function is invoked with the same inputs. While function caching forms the foundation for staging, it, by itself, is not sufficient to realize performance gains in practice. Typical events in discrete event simulators have time-varying, continuous inputs, which preclude matching function inputs between calls.

Staging significantly improves on function caching by introducing three techniques, called *function decomposition*, *refinement*, and *batching*. These techniques restructure computations such that their results are reusable even when a change in inputs would normally preclude reuse:

Function decomposition splits a large computation is split into several smaller sub-computations that are each dependent on only a subset of the inputs to the original computation. By carefully choosing the decomposition, we can reduce or eliminate the dependency on frequently varying inputs. For example, replacing a function $f(x, y, t)$ with an equivalent, decomposed version $f'(g(x, y), t)$ can allow $g(x, y)$ to be cached and reused even when the parameter t varies between calls.

Refinement further expands the applicability of function caching by taking advantage of the continuity of the physical model underlying the computation. When a small change in inputs is expected to lead to little or no change in the computed results, computing bounds then refining them to precise results can be more efficient than computing the same result from scratch. For instance, computing upper and lower bounds on node mobility may allow the simulator to eliminate costly computations to determine neighborhoods. In this case, the upper and lower bounds are computed such

that they are valid for a range of inputs and so can be cached and reused even when inputs vary slightly between calls.

The third staging technique, *batching*, reorders the computations within the simulator so that many independent, fine-grained computations can be executed more efficiently in a single pass. Function decomposition and refinement both transform the event stream in a simulator into an equivalent, but much finer grained, sequence of computations. Many of these computations are not time dependent, and so can be reordered without affecting simulation accuracy. Batching groups related computations together, and replaces them with a single computation which computes all the needed results efficiently in a single pass. Batching not only allows the utilization of more efficient global algorithms instead of independent local computations, but can also improve processor and memory cache performance by improving locality.

Staging fundamentally involves a space-time trade off. For staging to be worthwhile, the target computation must be more expensive than the cost of storing and fetching cached results from a potentially large table. Additionally, the cached results will likely increase the amount of memory required for the simulation, due to the cost of storing the cached results. Although this increase in memory use may increase virtual memory paging by increasing the working set, it may conversely reduce the working set by eliminating memory intensive computations.

The remainder of this paper illustrates the use of staging in a widely used network simulator under typical usage scenarios. We give examples of existing, ad hoc applications of staging in current state of the art simulators, identify new opportunities for staging, and evaluate the effectiveness of both intra- and inter-simulation staging in a ubiquitous and mature network simulation engine.

3 TRADITIONAL WIRELESS SIMULATION

Efficient and scalable wireless network simulators are critical to network research, but present unique challenges in their implementation. They differ from other simulators in several key ways, each of which introduces redundant computation at runtime. As a result, many commonly used wireless simulators are slow and do not scale gracefully with network size.

The fundamental reason redundant computation is prevalent is that wireless mobile networks have highly dynamic characteristics, which imply that simulation state must be recomputed dynamically and often. As nodes move about a simulated field, the network-level topology may change rapidly. Link characteristics, routing information, and network topologies must be maintained and recomputed during the simulation, and mobile nodes must continually update their positions in order to provide accurate information to the network model. In addition, complex physical models

make wireless simulation expensive. Since wireless is a broadcast medium, a straightforward simulation approach treats the network as a single broadcast LAN, incurring $O(n^2)$ run time in a network with n active nodes.

Existing wireless network simulators address some of the challenges of wireless networks. These range from general-purpose simulators, such as ns2 and OpNet (Chang 1999), to special-purpose and custom simulators including SWIMNet (Boukerche et al. 1999), MobSim++ (Liljenstam, Rönngren, and Ayani 2001), DaSSF (Liu et al. 2001), and GloMoSim (Zeng, Bagrodia, and Gerla 1998). These simulators have widely varying designs, including parallel or distributed event engines and specialized language features. Distributed simulators achieve scalability and performance by recruiting multiple simulator hosts. Even in such systems, each simulator host may perform a large amount of redundant computation that can be eliminated to improve efficiency.

We chose to study wireless simulation in the ns2 network simulator because it is widely used in academic research, and because it has a well-established and validated set of protocols. The protocol implementations in ns2 total over 150,000 lines of code, and provide accurate models for node mobility, wireless energy consumption, radio propagation and MAC-layer protocols.

ns2 tends to be slow and scale poorly with increasing number of nodes. As we show in the following sections, staged simulation can drastically reduce the amount of work required to simulate a wireless system by reducing redundant computation. These results are not specific to ns2, but can be applied likewise to more advanced simulation engines as well.

4 STAGED SIMULATION IN NS2

In the baseline ns2 implementation, the wireless physical layer and mobility models are the largest consumers of processing time in typical simulation scenarios. These components pose the most significant bottlenecks to efficiency and scaling. Consequently, we focus on staging computations related to node mobility and the wireless physical layer.

We incrementally describe four different types of staging, each employing a different approach to eliminating redundant computation. The first is an example of reusing common intermediate results across function calls. The second demonstrates the use of restructuring to enlarge the overlap in computation across calls. The third optimization illustrates precomputation as a staging technique, and the final one demonstrates inter-simulation staging by reusing results across multiple runs of the simulator.

4.1 Grid-Based Neighborhood Computation

For staging to be effective, redundant computations need to be readily identifiable. The monolithic structure of the default ns2 implementation, however, obscures the redundant computations it performs at runtime. Specifically, ns2 in particular, and wireless network simulators in general, perform numerous calculations to ultimately determine the set of nodes that will receive a given packet. These calculations depend on the positions of sending and receiving nodes, packet transmission and detection power levels, geography, and radio and antenna models. We note that many of these inputs will be identical or similar across computations, and show in Section 5 that the resulting redundant operations are significant and lead to non-linear scaling with network size.

To expose parts of this redundancy, we first apply a very simple grid-based staging approach where we reuse previously computed power levels for nearby nodes. We first divide the coordinate space into a grid of buckets, with each bucket holding a list of nodes positioned within the corresponding grid rectangle. This data structure can then be used to quickly determine if a group of nodes falls entirely outside the possible transmission range of a node, thereby eliminating the need to perform individual calculations for each node. Nodes in the remaining buckets, which may or may not be in range, are checked individually as before. In order to maintain the grid as nodes move during the simulation, we compute all of the times at which a node will cross a grid boundary, scheduling events at these times to update the grid as needed.

While grid-based decomposition in simulators is not novel, it serves as an initial application of staging that enables us to identify and eliminate other redundant applications through more advanced applications of staging in the subsequent sections. Nevertheless, grid-based neighborhood computation employs staging in two distinct ways. First, by grouping nodes into buckets, the simulator can reuse a single computed result for all nodes within the bucket. Furthermore, since the grid data structure will remain fixed across many packet transmissions, we can share and reuse a single global grid structure. We assume here, as is typical in ad hoc network research, that all nodes use uniform and constant transmission and reception parameters. This assumption does not present a limitation of the staged simulation approach, but simplifies our examples considerably.

4.2 Neighborhood Caching

Variations on the grid approach allow more advanced applications of staging using auxiliary computations to reduce redundancy in computation across packet transmissions. In typical simulation scenarios, inter-packet spacing is very

short in comparison to the speed at which nodes move. Depending on node mobility and traffic patterns, many hundreds or thousands of packets may be transmitted from a single node before nodes move a significant distance. That is, we should expect the inputs to, and hence the results of, the neighborhood computation for a node to be reusable across many packet transmissions.

Since inputs will vary slightly, we should not expect the neighborhood set to be identical to that computed during the previous packet transmission. However, a conservative upper-bound, or superset, of the neighborhood set will remain valid for some time after it is computed, depending on the amount of node mobility and the tightness of the bound. This holds similarly for a lower-bound or subset of the neighborhood set. We therefore restructure the neighborhood set computation to first compute upper and lower bounds on the result, then refine these bounds into an exact result. After restructuring the computation, intra-simulation staging is used to cache and reuse the common intermediate results, the two bounds, across many packet transmissions.

This restructuring introduces one additional parameter, Δt , to control the caching policy. This parameter fixes the desired epoch duration for which the bounds on the neighborhood set will be valid. If s_{max} is the maximum possible node speed in the movement scenario, then the maximum change in distance between two nodes in an epoch is just $\Delta r = 2s_{max}\Delta t$. If two nodes are within distance $r - \Delta r$ at some time, then they will remain within range r for Δt seconds into the future. Similarly, nodes beyond distance $r + \Delta r$ need not be considered at all for Δt seconds into the future.

We maintain a cache to capture the upper and lower bounds on the neighborhood set of each node. At most one cache entry is maintained for each node in the network. A cache entry, illustrated in Figure 1, is composed of an expiration time and two sets, $N_{r-\Delta r}$ and $N_{r+\Delta r}$, containing lists of the nodes within a ball of radius $r - \Delta r$ and those in the annulus with radii $r \pm \Delta r$. During packet transmission, the cache manager computes the set of nodes within range of a given node by first looking for a valid cache entry. Finding an entry that has not yet expired, it can immediately consider all nodes in the list $N_{r-\Delta r}$ to be within range. The second list $N_{r+\Delta r}$ is then scanned, and each node found to be within range is appended to the final result. At the same time, it can cheaply but conservatively update the lists, moving some nodes from $N_{r+\Delta r}$ to $N_{r-\Delta r}$ and eliminating others from $N_{r+\Delta r}$ entirely. If, on the other hand, no cache entry is found during packet transmission, the cache manager consults the underlying mobility (grid) manager and constructs a cache entry with expiration Δt seconds into the future.

In the above caching scheme, there is some additional overhead during cache misses, when computing $N_{r\pm\Delta r}$, since a larger radius is considered than previously necessary.

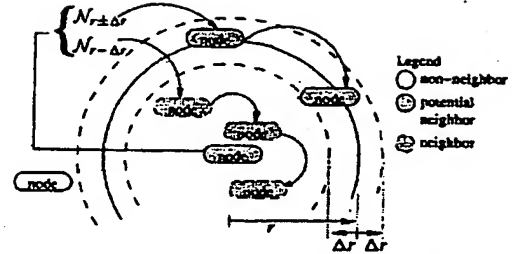


Figure 1: Computing Bounds on Node Movement Enables the Simulator to Examine Only the Nodes Located in an Annulus $N_{r\pm\Delta r}$ During Packet Transmission by Node at Center

This overhead is controlled directly with the parameter Δt , which fixes the longevity and the accuracy of cache entries. In addition, there is overhead associated with scanning the list of nodes in $N_{r\pm\Delta r}$ during each cache hit, but this is also limited by appropriately choosing the Δt parameter. We analyze these overheads in Section 5.

4.3. Perfect Caching

There is a large overlap in computation when constructing cache entries for nodes using the neighborhood caching scheme. We use precomputation to address this redundancy by computing many cache entries simultaneously. When constructing a cache entry, a node normally examines all nodes within a potentially large radius. If many nodes in a reasonably dense network are active, and each periodically construct cache entries on-demand and independently, each pair of nodes will eventually be considered twice.

A staged simulation approach, which we term *perfect caching*, eliminates redundancy by precomputing all cache entries simultaneously. This approach maintains the same data-structures as neighborhood caching. But, rather than calculating cache entries on-demand, it precomputes all cache entries at the beginning of every Δt epoch. All normal queries for neighborhood information are then guaranteed to hit the cache. There are several possible advantages to precomputation. First, we only need to examine each pair of nodes at most once, rather than twice, to compute all of the entries. Second, the positions of all nodes can be updated a single time at the start of the generation process. Previously, it was necessary to update the positions of all nodes within range of the sender during each cache miss. Finally, memory locality should improve when precomputing all entries simultaneously as compared to individually on-demand.

The overhead of this technique is a scheduled event during each Δt epoch, and possibly some wasted computation if some nodes do not send packets during an epoch, and thus do not use their cache entries. In a sparse or

quiet network, perfect caching might construct more entries than needed during the simulation. This problem can be addressed directly by appropriately choosing the Δt epoch parameter.

4.4 On-Disk Caching

A final inter-simulation staging application improves on perfect caching, and demonstrates how staging can be applied across multiple similar runs of the simulator. The intra-simulation examples above reduce the amount of computation significantly, but also add some additional events to the event queue leading to more work in the event scheduler. Event queue management is a well-studied problem, especially in the particular case of the Calendar Queue used in ns2. However, we can eliminate the work done by many events by looking at a set of simulation runs together. This application of inter-simulation staging therefore builds on the previous optimizations by reducing the number of scheduled events generated by the grid manager and the cost of constructing neighborhood cache entries in the perfect caching scheme.

First note that, by itself, perfect caching generates strictly more events than the on-demand caching approach, and may actually compute results that are not used in any particular simulation run. But, also observe that perfect caching will perform identical work during multiple simulation runs using the same mobility scenario. In the second and subsequent runs of the simulator we can eliminate these extra events, as well as most cache maintenance, by writing all cache entries to disk every Δt seconds during the first simulator run. Subsequent runs can obtain cache entries from disk rather than maintaining an underlying cache manager or grid. This technique then introduces two phases. The *generation-phase* is identical to perfect caching except that all cache entries are spooled to disk. The *use-phase* does not maintain a grid, does not need to track changes to node positions, and requires no scheduler events. Instead, cache entries are read from disk serially as needed during packet transmission. A set of runs with the same mobility model will use the more expensive generation-phase for the first run, and the less expensive use-phase for all remaining runs.

5 EVALUATION

We have implemented each of the optimizations detailed in Section 4 in the ns2 simulator. We find that even the simplest application of staging reduces the run time of the simulator significantly, and allows for practical simulation of much larger network sizes than previously feasible. We show that more advanced intra-simulation techniques improve stability and robustness of the simulator, while the application of inter-simulation staging improves performance yet further.

With the latest staged implementation, we regularly simulate networks of over 1000 nodes in the time it previously took to simulate networks of hundreds of nodes.

In addition to evaluating total simulation run time using our techniques, we also characterize the effect of each parameter we have introduced. For staging to be effective, it must be possible to easily or automatically find near-optimal choices for these parameters and, at the very least, avoid parameter choices that would lead to run time behavior worse than the default, non-staged implementation. We first describe our test environment and changes required to add staging to the simulator, then present the results of our staging techniques.

5.1 Evaluation Platform and Environment

We take as our baseline a modified ns2 version 2.1b9a simulator. All simulations were completed on a single-processor machine equipped with 1.7GHz Pentium 4 processor and 256MB of physical memory. Physical memory is an important constraint in ns2; more generous machines can simulate proportionally larger networks before becoming memory-limited. Before implementing our staging techniques, we made a few non-standard modifications to improve the baseline ns2 code. Most notably, we disabled all unused packet headers to reduce packet sizes and improve memory locality, and implemented more efficient packet tracing. This improved run time by 85% for a 250 node network. The performance results detailed in this paper are computed relative to this optimized ns2 baseline implementation.

Staging can impact the performance of a simulator by introducing fine-grain events and changing the event distribution observed by the event scheduler. Calendar queue schedulers are particularly sensitive to such perturbations (Oh and Ahn 1999). To counteract the sensitivity of the calendar queue scheduler to the event distribution, we modified the calendar queue event scheduling algorithm to re-optimize the event queue after 30 seconds of simulated time, effectively avoiding occasional mis-predictions by the scheduler.

Overall, our simulation runs closely resemble those discussed in Broch et al. (1998), a very common setup. We used standard CMU Monarch mobility and communication model generators from the standard ns2 distribution. As an exemplar of typical wireless network research, we chose the AODV ad hoc routing protocol implementation included with ns2. Our results are not specific to these choices of application, mobility model, or communication pattern. These system parameters, summarized in Table 1, closely follow the standard values used in ad hoc networking literature. Although the nominal reception radius for our antenna model is only 250 meters, we use the transmission detection radius of 351 meters for all optimizations in order to properly account for interference effects.

Table 1: Default Simulation Parameters for Experiments

Network load	
model	Constant bit rate
concurrent data streams	30
packet size & rate	512 bytes \times 8 packets/s
Node mobility	
model	random-waypoint
maximum node speed	5 m/s
pause time	10 s
field density	≈ 31 nodes / km ²
Simulation	
routing protocol	AODV
simulation time	400 s

5.2 Simulator Performance

We first examine how the different applications of staging affect total simulation execution time using a 1000 node network. In this experiment, we fix grid granularity at 250 meters and Δt at 2 seconds, and later describe their selection and the sensitivity of staging to these parameters. We run our simulations with various applications of staging enabled, as shown in Table 2. For each level of staging, we run the simulator on five randomly generated networks and present the average of the execution times. The sample standard deviation for each data point is less than 0.2%.

Table 2: Levels of Ns2 Optimization for Experiments

Level	Optimizations
L ₀	Ns2 baseline: improved tracing and packet size
Intra-simulation staging	
L ₁	L ₀ + Grid-based
L ₂	L ₁ + Caching
L ₃	L ₂ + Perfect caching
Inter-simulation staging	
L _{4a}	L ₃ + On-disk caching (generation)
L _{4b}	L ₃ + On-disk caching (use)

The speedup achieved by increasing levels of staging relative to the baseline simulator is shown in Figure 2. These results, obtained using a 1000 node network, highlight especially the benefits of the simplest intra-simulation staging L₁ technique and of the inter-simulation staging technique. Optimization level L_{4b}, the second phase inter-simulation staging approach, improves simulation run time significantly in comparison to using only intra-simulation techniques. Also, the one-time cost of the first phase, L_{4a} is no worse than the best possible intra-simulation technique L₃. Thus, in this case inter-simulation staging imposes no additional cost during the first run of a series, but offers a significant speedup during subsequent runs.

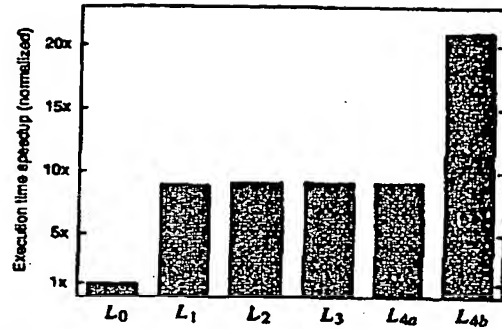


Figure 2: Speedup in Execution Time with Increasing Staging Relative to Baseline Ns2 Implementation using a 1000 Node Network

5.3 Scaling with Network Size

In order to evaluate how staging affects simulation scale, we simulated networks with varying number of nodes while holding the application-level load constant and increasing the field size to maintain a constant node density.

Figure 3 shows that staging can improve the scalability of wireless simulators by reducing redundant computations. This experiment also demonstrates the benefits of inter-simulation staging, which achieves 56% improvement over the intra-simulation staging techniques in 1000 node networks. Although the different intra-simulation staging approaches show similar performance in this experiment, they exhibit different behaviors as optimization parameters or network characteristics change. As we show in the next two sections, the more advanced optimizations offer increased robustness and stability, an advantage not evident in Figure 3.

Additional experiments indicate similar performance benefits using networks of varying density, up to more than twice the density used above. Very dense networks, however, expose a trade-off in our disk-based inter-simulation optimization. In our implementation, cache entries are stored on disk during the first simulator run, and must be read from disk and processed during each subsequent run. While most of these disk accesses are easily pipelined and dispatched in the background, there is still a non-negligible CPU cost for dispatching and processing data stored on disk. As network density increases, the cache entries grow larger and cache processing may become more expensive than simply recomputing results from in-memory data.

This trade-off is present to some extent in any result caching scheme, and designers must be careful that cache overhead is less than the cost of recomputation. But in practice we find that only the disk-based caching optimization might impose a significant processing overhead, for certain

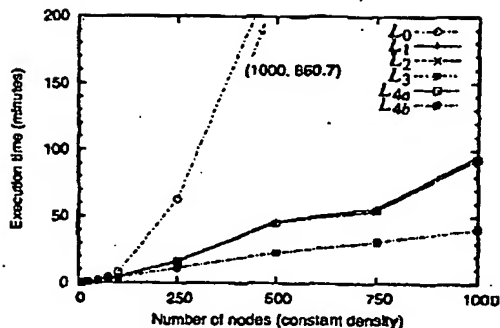


Figure 3: Effect of Network Size on Total Simulation Run Time Holding Node Density Constant

networks, and that the optimization offers a net improvement in performance for networks of reasonable density.

5.4 Optimization Parameters

It is important to characterize the effect of any new simulation parameters introduced by our optimization techniques. We study simulation performance under various choices for optimization parameters and examine the robustness and stability of the different optimization levels. Recall that the grid-based intra-simulation approach introduces a granularity parameter, and the caching intra-simulation approach a Δt lookahead parameter.

We first evaluate the effect of varying grid granularity on each level of staging. Intuitively, it is clear that a very fine granularity will give rise to many grid-crossing events as nodes move about in the topology, and also leads to more work in packet transmission, as many empty bins will be scanned for nodes. Conversely, a very coarse granularity reduces to a single bucket and, essentially, a scan over all nodes during each packet transmission or cache miss. A reasonable choice is to use the node transmission radius, which requires a scan of roughly nine buckets during each transmission or cache miss.

We run the simulator on a single 250 node network with the same configuration as before and Δt fixed at 2 seconds, but vary the grid granularity. Figure 4 verifies our intuitive description of the effects of grid granularity. Interestingly, we find that any choice of granularity other than the two extremes yields a substantial improvement in run time under L_1 staging, with only minor variation between 500 and 2000 meters, with the optimum choice approximately 1500 meters.

In this experiment, even the right-most extreme performs much better than the ns2 baseline implementation since we avoid creating events and copies of the packet for nodes outside the transmission range. Further, much of the degradation due to a poor choice in granularity is mitigated

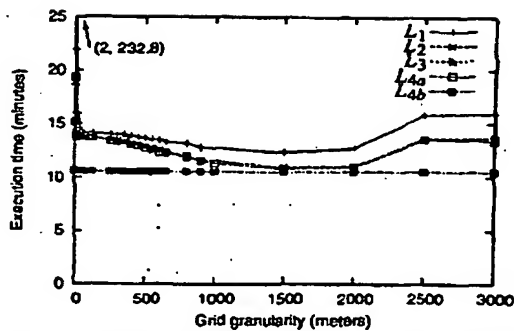


Figure 4: Effect of Varying Grid Granularity on Simulation Run Time

by the use of the higher levels of staging. In these cases, the poorly-tuned grid is consulted only in the rare case of a cache miss.

The choice of grid granularity depends on the particular choice of node mobility and load patterns. In practice, we find that the optimal choice of granularity can be as low as 250 meters, but is rarely higher than 2000 meters. In all cases we have examined, the trends are similar to those presented above, making automatic tuning a feasible approach.

5.5 Caching Lookahead Parameter

The overhead of constructing cache entries is controlled by the Δt parameter to the neighborhood caching routine. Recall that Δt specifies the desired expiration time when constructing a cache entry. A larger value means that a larger radius must be examined to build a cache entry, leading to a larger data structure, but allowing the cache entry to remain valid for longer. We set up our simulator as the previous experiment, but fix the grid granularity at 250 m.

Figure 5 shows how Δt controls the cache hit rate (top), and the sizes of the two neighborhood sets $N_{r-\Delta r}$ and $N_{r+\Delta r}$ stored in cache entries (bottom). We only show the results for L_2 caching; those for L_3 perfect caching and the first phase L_{4a} of intra-simulation staging are identical. For reference, the actual average neighbor set size for queries is shown as constant N_r .

The overheads associated with caching are limited by the cache hit rate and $N_{r+\Delta r}$. A very small value for Δt leads to many cache misses, each of which is potentially expensive. Conversely, a large value for Δt forces both cache hits and misses to process a larger set $N_{r+\Delta r}$. The cache is effective for reasonable values of Δt , roughly 2 to 4 seconds, with high hit rate but still reasonably sized $N_{r+\Delta r}$. The curves for the neighborhood set sizes can be explained geometrically based on the known transmission radius, and the average number of neighbors of transmitting nodes. The cache hit rate is a function of the average inter-

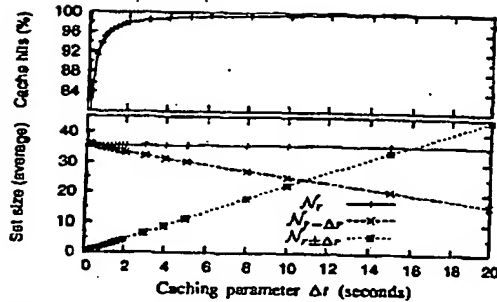


Figure 5: Effect of Varying Caching Parameter Δt on Cache Hit Rate and Neighborhood Sizes

packet spacing. While our implementation does not pick Δt automatically, the figure shows that a near-optimal value for parameter Δt can be computed as a function of the packet rate, node density, and transmission radius.

Surprisingly, even with such varying cache behavior there is very little overall change in total simulation run time. Further experiments indicate that over the entire range of values in Figure 5, run time varies by at most 5% over the range of Δt values shown. The L_2 , L_3 , and L_{4a} staging levels all perform similarly, while the second phase L_{4b} inter-simulation approach improves run time by approximately 30% as compared to L_3 , independent of the Δt parameter. As with the grid granularity, nearly any reasonable choice of parameter will work well for the highest levels of staging.

6 RELATED WORK

Several important examples of staging can be found in existing simulators. In our analysis of the ns2 implementation, we identified applications of staging, but find that the technique of staging is not widely applied in the implementation or recognized in the literature. There has been no prior recognition or development of the technique of staging as a general approach to simulation optimization.

The NixVector (Riley, Ammar, and Fujimoto 2000) approach improves wired-network routing efficiency in the ns2 simulator by computing and caching routes on demand rather than maintaining a complete routing table. This approach has not been applied between multiple runs of the simulator, nor does it eliminate redundancy when inputs vary slightly between computations.

A second example from ns2 is a grid implementation very similar to our L_1 staging. A key difference is that we expose and explore the parameter space of grid granularities, while the previous attempt uses a hard-coded granularity of 1 meter. In typical scenarios, this choice leads to performance worse than the baseline. Similarly, Wu and Bonnet (2002) propose an alternative packet transmission routine for ns2,

essentially equivalent to our L_1 staging with granularity parameter ∞ . Again, we have shown that this choice of granularity is particularly inefficient as compared to nearly any other choice. These examples illustrate the importance of properly characterizing staging parameters and relating them to system variables such as the transmission radius and expected number of neighbors.

In the context of discrete event simulators, we find occasional use of staging or similar techniques to improve performance. Splitting (Glasserman, Heidelberger, Shahabuddin, and Zajic 1996), cloning (Hybinette and Fujimoto 1997), and updateable simulations (Ferencsi et al. 2002) are three related techniques which eliminate identical computations in multiple runs of the simulator. These techniques do not exploit redundant computations within a single run of the simulator, nor do they address computations which are similar but not identical.

Boukerche et al. (1999) propose a two-phase design for Personal Communications System (PCS) network simulation using SWIMNet. This design is used to facilitate various lookahead optimizations in a parallel simulation engine, rather than to eliminate redundant computation or optimize multiple runs of the simulator.

A popular technique for improving scale and performance uses distributed simulation (for example Boukerche et al. 1999, Liu et al. 2001, and Liljenstam, Rönngren, and Ayani 2001), sometimes combined with specialized language features (for example Zeng, Bagrodia, and Gerla 1998). These approaches are complementary to our optimizations, since staged simulation can be applied equally well to both distributed and centralized designs. Other techniques are used to reduce simulation run time, such as model abstraction and approximation (Huang, Estrin, and Heidemann 1998, Gadde, Chase, and Vahdat 2002). Our approach differs from model abstraction in that we do not alter in any way the final result of computations. Additionally, abstraction may not be possible if the system of interest has not yet developed stable or well-understood models.

Finally, we note that staging as a concept is a general technique, employed most notably in compilers and iterative programming. Chambers (2002) discusses a staged compilation technique that combines partial precompiling of code coupled with dynamic optimizations at runtime. Iterative programming is a general framework for describing computation. Like staged simulation, it relies on reusing results, intermediate values, and extraneous values from previous iterations. Liu, Stoller, and Teitelbaum (1996) discuss methods for automatically extracting this information using program and data-flow analysis. We find this particular approach unsuitable for large and complex simulator implementations, where data-flow and simulation behavior depend very heavily on the particulars of a simulation run. Additionally, the use of multiple languages compounds the difficulty of low-level automatic program analysis.

7 CONCLUSIONS

We propose a general technique, termed staged simulation, for reducing the run time of discrete event simulators. The central idea is to eliminate redundant or partially-redundant computations typical in simulations by caching and reusing the results of computations. The technique consists of identifying redundant computation both within single runs as well as across consecutive runs of the simulator. Staging then relies on precomputing, caching and reusing partial results to eliminate redundant computation. Our technique is general and applicable to a wide range of designs, including parallel and distributed simulation engines.

We show that staging is an effective technique for reducing simulation run time without loss of accuracy, and is effective in a wide range of simulation scenarios including varying mobility and communication patterns, network sizes, and node densities. We implement three levels of intra-simulation staging and one level of inter-simulation staging in the ns2 wireless networking simulation system. Simple intra-simulation optimizations are found to reduce simulator run time by a factor of 9 and to improve simulator scalability from networks of hundreds of nodes to networks of ten thousand nodes. An application of inter-simulation staging can reduce run time even further to a factor of 21 over the non-staged implementation. We find that the techniques are robust in the choice of parameters, and these parameters appear easy to estimate automatically as a function of other simulation variables and observed runtime behavior.

REFERENCES

- Boukerche, A., S. Das, A. Fabbri, and O. Yildiz. 1999. Exploiting model independence for parallel PCS network simulation. In *Workshop on Parallel and Distributed Simulation*, 166-173.
- Broch, J., D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. 1998. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *ACM/IEEE Intl. Conference on Mobile Computing and Networking*, 85-97.
- Chambers, C. 2002. Staged compilation. *ACM SIGPLAN Notices* 37 (3).
- Chang, X. 1999. Network simulations with OPNET. In *Winter Simulation Conference*, ed. P. Farrington, H. Nembhard, D. Sturrock, and G. Evans, 307-314. Piscataway, NJ: IEEE Press.
- Ferenci, S., R. Fujimoto, M. Ammar, K. Perumulla, and G. Riley. 2002. Updateable simulation of communications networks. In *Workshop on Parallel and Distributed Simulation*, 107-114.
- Gadde, S., J. Chase, and A. Vahdat. 2002. Coarse-grained network simulation for wide-area distributed systems. In *Communication Networks and Distributed Systems Modeling and Simulation Conference*.
- Glasserman, P., P. Heidelberger, P. Shahabuddin, and T. Zazic. 1996. Splitting for rare event simulation: Analysis of simple cases. In *Winter Simulation Conference*, ed. J. Charnes, D. Morrice, D. Brunner, and J. Swain, 302-308. Piscataway, NJ: IEEE Press.
- Huang, P., D. Estrin, and J. Heidemann. 1998. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. In *MASCOTS*, 241-248.
- Hybinette, M., and R. Fujimoto. 1997. Cloning: a novel method for interactive parallel simulation. In *Winter Simulation Conference*, ed. S. Andradóttir, K. Healy, D. Withers, and B. Nelson, Piscataway, NJ: IEEE Press.
- Liljenstam, M., R. Rönngren, and R. Ayani. 2001. MobSim++: Parallel simulation of personal communication networks. *IEEE DS Online* 2 (2).
- Liu, J., L. Perrone, D. Nicol, M. Liljenstam, C. Elliott, and D. Pearson. 2001. Simulation modeling of large-scale ad-hoc sensor networks. In *European Simulation Interoperability Workshop*.
- Liu, Y., S. Stoller, and T. Teitelbaum. 1996. Discovering auxiliary information for incremental computation. In *ACM SIGPLAN-SIGACT Symposium*, 157-170.
- Oh, S., and J. Ahn. 1999. Dynamic calendar queue. In *Annual Simulation Symposium*.
- Riley, G. F., M. H. Ammar, and R. Fujimoto. 2000. Stateless routing in network simulations. In *MASCOTS*, 524-531.
- Royer, E., and C. Toh. 1999. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications* 6 (2): 46-55.
- The VINT Project. 1995. Ns-2 network simulator. Available at: <<http://www.isi.edu/nsnam/ns>> [accessed July 1, 2003].
- Wu, S., and C. Bonnet. 2002. An alternative packet transmission procedure for mobile network simulation. In *Intl. Symposium on Performance Evaluation of Computer and Telecommunication Systems*.
- Zeng, X., R. Bagrodia, and M. Gerla. 1998. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, 154-161.

AUTHOR BIOGRAPHIES

KEVIN WALSH is currently a Ph.D. candidate in Computer Science at Cornell. His e-mail address is <kwalth@cs.cornell.edu>.

EMIN GÜN SIRER is currently an Assistant Professor in Computer Science at Cornell. His e-mail address is <egs@cs.cornell.edu>.

Fault-tolerant Grid Services Using Primary-Backup: Feasibility and Performance

Xianan Zhang*
xzhang@cs.ucsd.edu

Dmitrii Zagorodnov*
dzagorod@cs.ucsd.edu

Matti Hiltunen†
hiltunen@research.att.com

Keith Marzullo*
marzullo@cs.ucsd.edu

Richard D. Schlichting†
rick@research.att.com

Abstract

The combination of Grid technology and web services has produced an attractive platform for deploying distributed applications: Grid services, as represented by the Open Grid Services Infrastructure (OGSI) and its Globus toolkit implementation. As the use of Grid services grows in popularity, tolerating failures becomes increasingly important. This paper addresses the problem of building a reliable and highly-available Grid service by replicating the service on two or more hosts using the primary-backup approach. The primary goal is to evaluate the ease and efficiency with which this can be done, by first designing a primary-backup protocol using OGSI, and then implementing it using Globus to evaluate performance implications and tradeoffs. We compared three implementations: one that makes heavy use of the notification interface defined in OGSI, one that uses standard Grid service requests instead of notification, and one that uses low-level socket primitives. The overall conclusion is that, while the performance penalty of using Globus primitives—especially notification—for replica coordination can be significant, the OGSI model is suitable for building highly-available services and it makes the task of engineering such services easier.

1 Introduction

A Grid infrastructure, being a collection of resources, is prone to many kinds of failures: application crashes, hardware faults, network partitions, and

unplanned resource downtime. Most Grid platforms have mechanisms for tolerating at least some kinds of these failures. These mechanisms typically retry failed executions, perhaps starting at a recent checkpoint [13, 4]. Furthermore, the need for fault tolerance in Grid infrastructures is well known; an overview of these techniques and a unifying failure handling framework, called *Grid Workflow*, is given in [12].

Recently, however, the emphasis in the Grid standardization efforts has moved from a focus on supporting job execution to service-oriented architectures that can be used not only for the traditional resource-intensive scientific computation tasks, but also as a general distributed computing platform. Specifically, the recent GGF (Global Grid Forum) standards define Grid computing platforms as collections of *Grid services* [9]¹ that include services that provide the Grid computing infrastructure, e.g., scheduling, and monitoring, as well as services that comprise the Grid application itself. While the fault-tolerance issues have been extensively explored in the traditional job execution scenarios, the issue of handling failures in the Grid services model as represented by the Open Grid Services Infrastructure (OGSI) [10] and its Globus [8] toolkit 3 (GT3) implementation, remains largely unexplored. Given that Grid services are expected to become the basis for commercial as well as scientific applications, such support is critical for wide-scale acceptance.

This paper addresses the problem of building highly available Grid services by replicating a service on two or more hosts. Making services highly available is not a new research area: it has been a research topic for decades, and there are commercial products for making non-Grid services highly available. So, one

*Department of Computer Science and Engineering, University of California, San Diego

†AT&T Research Lab

¹a new standard, Web Service Resource Framework (WS-RF) works on the convergence of web services and grid services.

might assume that some standard approach could be used for Grid services. Which approach to use, however, requires some study:

1. While there are currently only a few examples of Grid services, a common theme is that such services are expected to be *stateful*. This is in contrast to the closely related technology of Web services, which are *stateless* in that any changes to the service's state that must persist across failures is recorded in a database. If the machine executing a Web service fails, then the client can rebind to another machine (perhaps via a load balancer) that can reference the persistent state in the database². Commercial products, like the Veritas cluster manager [16], also assume that a failed server can be recovered by restarting it on another machine: any persistent state is kept in files or a database. A service that is stateful can have a lower latency (by avoiding writing state to a database) and have a simpler client-server protocol.
2. Extrapolating from existing services that are part of Grid systems, we can expect some of the services to have nondeterministic behaviors. A nondeterministic service by itself offers no problems to a client. Maintaining the consistency among replicas of a nondeterministic service is a problem, though: two replicas may become inconsistent even though they execute the same sequence of commands.

Nondeterminism can arise from many sources. At one end of the spectrum, a service's methods may be explicitly nondeterministic. For example, it is often better to randomly choose a "good" resource than to deterministically choose the "best" resource, either because it is computationally easier to do or because doing so spreads the load among several resources [3]. At the other end of the spectrum, nondeterminism may arise from timing issues, such as when exactly an external event (like an interrupt) occurs. For example, a resource manager service may use a lease-like mechanism to reclaim resources: if, after an interval of time, the use of a resource is not reconfirmed, then the manager automatically reclaims the resource. Consider a resource management service with two replicas where a client requests a resource of some class C . Let there be two resources c_1 and c_2 of class C , and c_1 has been previously allocated. One replica may get the client's request before the

lease for c_1 has expired, and so allocate c_2 to the client. The second replica may instead get the request after the lease has expired and allocate c_1 to the client.

3. To make service integration easier, Grid services are designed using very high-level protocols and services. All else being equal, one would not expect that a service built on top of, say, SOAP will have the same latency as a service built directly on top of TCP. Using the OGSI functions to provide for high service availability is appealing: one could provide it as a feature portable across any OGSI service. But, if the performance is much worse than the same feature implemented at a low level, then performance may outweigh the engineering appeal of a high-level implementation.

The first two points suggest using a *primary-backup approach* for service availability. Point one requires replicas to be consistent with each other. For example, a client could interact with replica r_1 , and then start interacting with another replica r_2 if r_1 fails. Anything that the client knows from r_1 about the state of the service should also be known by r_2 . Point two is addressed by having only one replica, the primary, respond to requests. The primary will keep one or more backup services consistent and ready to take over should the primary fail [6].

The primary goal of this paper is to evaluate the tradeoffs associated with using primary-backup as a fundamental technique for building highly available Grid services in the context of OGSI and Globus. Much of this focuses on the third point above—the tradeoff of performance versus use of facilities provided by the OGSI standard. We first designed a primary-backup protocol using OGSI to determine whether it supplies the necessary features, such as state update and client rebinding, and to see what changes might be needed to support such an approach. As described in Section 2, we found that it is not hard to accommodate primary-backup, and that the solution is simple and requires only small changes to the service to handle non-determinism. The use of the OGSI notification interface to handle replica updates is perhaps the key distinguishing feature of this approach.

We then implemented this approach using GT3 to better understand the performance implications and tradeoffs of doing primary-backup at such a high level. In particular, using a simple example Grid service, we compared the performance of this notification-based approach to variants in which replica update is done using standard Grid service method calls and TCP, respectively. Our example Grid service implements a sim-

²Some state, like a "shopping basket", can be bound to an individual server. Losing such state due to a server failure isn't often seen as being critical.

```

var target // points to the current primary
var ops // contains records of on-going operations

INIT_CLIENT ( ) // called when client binds to a Grid service
  (replica1, replica2, ... replican) ← find_replicas();
  target ← replica1;
  ops ← ∅;
  for each host ∈ {replica2...replican} do
    register_notification(&FAILURE_HANDLER, host,
      FAILURE);
  stub.init();

OP (params)
  var op // object for holding operation parameters
  var done // a semaphore for waiting until success
  var result // object for holding results of executions

  op ← { params, &done, &result };
  ops ← ops ∪ &op; // add op to the list of on-going operations
  create_thread(&INVOKE_OP, &op);
  wait_on_semaphore(&done); // wait for someone to succeed in op
  ops ← ops \ &op; // remove op from the list
  return result;

INVOKE_OP (op)
  var result // object for holding results of executions
  result ← stub.op(op.params); // make the SOAP call
  if result ≠ failure then
    op.result ← result; // pass result back to OP()
    signal(op.done); // wake it up

FAILURE_HANDLER (new_primary)
  target ← new_primary;
  for each op ∈ ops do
    create_thread(&INVOKE_OP, op);

```

Figure 2. Pseudocode for the fault-tolerant stub on the client.

failover duration, we don't wait for the *stub.op()* to return (it may take a while for the TCP socket to time out), so we re-submit the request to the new primary in *FAILURE_HANDLER* by spawning another *INVOKE_OP* thread. The parameters for this invocation are kept in the list *ops*. Eventually, some invocation should succeed, allowing *OP* to wake up and return the result.

The OGSi model specifies a method for clients to deal with failures of Grid service instances. Specifically, each Grid service has a persistent handle called a GSH (Grid Service Handle) and this handle can be resolved into a handle, called a GSR (Grid Service Reference), for an instance of this Grid service. The GSR may become invalid over time and the client can reacquire a valid GSR by re-resolving its GSH. The handle resolution is performed by a Grid service called the Handle Resolution Service. Although our design could incorporate this approach, in this paper we use a design that by-passes the Handle Resolution Service for two reasons:

- The Handle Resolution Service, if not fault-tolerant itself, would provide a single point of failure that could make all Grid services that rely on it unavailable.

```

var rate_sending // time interval for sending heartbeats

INIT_PRIMARY ( )
  claim_notification_source(HEARTBEAT); // register as source
  claim_notification_source(STATE.UPDATE);
  schedule(&HEARTBEAT_GENERATOR, rate_sending); // run
  regularly

HEARTBEAT_GENERATOR ( )
  notify_change(HEARTBEAT); // send a notification

EXECUTE (request)
  var result // object for holding results of executions
  result ← check_previous_requests(request);
  // if request is completed result is not NULL, but for new requests it
  is
  if result = NULL then
    var state // encoding of application state
    result ← service.op(request.params); // execute the request
    state ← service.extract_state(); // obtain state of the
    application
    notify_change_with_ack(STATE.UPDATE, state); // waits for
    acks
  return result;

```

Figure 3. Pseudocode for the primary service.

- In the handle resolution approach, the client only detects the failure of the primary when it attempts to use its GSR. In our approach, the client is notified immediately.

The code on the replicas is interposed between the Grid infrastructure and the service implementation. For each client *OP* there is an implementation of that operation on the server. To make stateful primary-backup replication possible, the service must implement two additional methods for state transfer: *extract_state()* and *inject_state()*. Ideally, state transfer can be done by a small set of values describing all the relevant application state, but in the extreme it could be a full application checkpoint.

On the primary, as shown in Figure 3, we intercept each one of the operations with the *EXECUTE* method. It first checks whether this request has already been processed—this can happen when a server crashes after sending the state to the backups, but before replying to the client. In that case the old result is returned without executing the request. Otherwise, the request is executed, followed by the extraction of state, which is sent to backups via notifications. Note that *notify_change_with_ack* blocks until it gets an acknowledgment from every backup. In the initialization routine, the primary advertises itself as a source of two types of notifications (*HEARTBEAT* and *STATE.UPDATE*) and schedules a heartbeat routine to run regularly.

Figure 4 shows the pseudocode for backups. During normal operation they receive two kinds of no-

```

var rate_checking // time interval for checking for notifications
var last_notification // timestamp of the last notification
var primary_is_up // boolean flag
var senior // this is the senior backup

INIT_BACKUP ()
  (replica1, replica2, ... replican) ← find_replicas();
  primary_is_up ← TRUE;
  if my_url() = replica2 then
    senior ← TRUE;
    register_notification(&HB_HANDLER, replica1, HEARTBEAT); // register sinks with primary
    register_notification(&STATE_HANDLER, replica1, STATE.UPDATE);
    claim_notification_source(FAILURE); // register as a source for clients
    schedule(&FAILURE_DETECTOR, rate_checking);
    SETUP_SENIOR(replica2);

SETUP_SENIOR (senior_url)
  if senior = TRUE then
    claim_notification_source(HEARTBEAT);
    claim_notification_source(STATE.UPDATE);
  else
    register_notification(&HB_HANDLER, senior_url, HEARTBEAT);
    register_notification(&STATE_HANDLER, senior_url, STATE.UPDATE);

FAILURE_DETECTOR ()
  if (current_time() - last_notification) > rate_checking then
    if senior = TRUE then
      switch_to_primary();
      notify_change(FAILURE); // notify client
    else
      if primary_is_up = TRUE then
        primary_is_up ← FALSE; // wait for the next timeout
      else
        INIT_BACKUP(); // backups re-initialize, electing a new primary
  else
    if primary_is_up = FALSE then
      primary_is_up ← TRUE;
      (replica1, replica2, ... replican) ← find_replicas(); // elect new senior
      SETUP_SENIOR(replica2);

STATE_HANDLER (state)
  service.inject_state(state);
  last_notification ← current_time();

HB_HANDLER ()
  last_notification ← current_time();

```

Figure 4. Pseudocode for the backup service

tifications: their STATE_HANDLER receives state updates and injects the state into the service application and their HB_HANDLER receives heartbeats. Both store the current timestamp in the global variable *last_notification*. FAILURE_DETECTOR checks this variable to make sure it is not stale. If it is then the primary is assumed to have failed.

Switching to a new primary can take a long time because it needs to register as a source of notifications and all backups must re-bind to the new primary. If we delayed client-bound failover notification until re-binding is complete, the failover time of our system would be extremely large (binding can take seconds!). We avoid this performance penalty by binding all backups to one special backup, which we call the *senior* backup, at the time of service initialization.

If a failure is detected, the senior backup becomes the primary and notifies the client immediately, since it already has all backups registered with it to receive

state updates and heartbeats. The remaining backups then chose a new senior and bind to it “off-line”, without delaying processing of client requests. This binding is implemented in the SETUP_SENIOR method, which is called during initialization and during recovery. In the rare situation that the senior backup fails together with the primary, all surviving backups will assume the failure of the senior after the second missing heartbeat and they will go through full re-initialization by calling INIT_BACKUP.

For simplicity, the pseudocode shows that the state is applied immediately by calling *inject_state* in STATE_HANDLER. In a real implementation it would be better to queue up the state update, send back an acknowledgment and apply the state later, so as to impose as small of a penalty on the response time as possible. As implemented, the protocol queues state updates and applies them later in this way. Doing so can slow down failover because the backup may have to apply queued

state messages before processing new requests.

3 Performance

While it appears that OGSi is a suitable platform for building primary-backup fault tolerance, the overhead of replication may ultimately determine whether the technique is useful in practice. In this section, we describe the performance of our prototype implementation using GT3.

Our example highly available Grid service is a simplified version of the well-known Condor Matchmaker service. We measured the transfer overhead, the request response time, and the failure notification overhead of a prototype service structured according to the primary-backup approach described above. We performed experiments on a pair of dual-CPU Pentium II 300MHz workstations with 400Mb of memory, running Linux 2.4. We only considered a system with a primary and one backup, since this is by far the most common way primary-backup is used.

3.1 Matchmaker Grid Service

We designed the Grid Matchmaker service based on existing (but more complex) non-Grid services, such as Condor Matchmaker [14], Java Market [2], and the resource management tools in Globus [7]. We chose to use this service because it is an example of an important class of Grid service, and because is inherently non-deterministic.

Our Matchmaker service keeps track of machines available in the Grid, accepts requests for allocating machines, and maps each request to a suitable machine. There are two kinds of requests: one is a *resourceAdvertise* request, and the other is a *jobSubmit* request. A *resourceAdvertise* request provides information about a machine that is available for allocation. The input of this request is: the resource ID, the available CPU speed, the available memory size, the available disk size, the machine's IP address, and an identification string used to implement a simple capability for using the machine. A *jobSubmit* request sends a specification for a desired machine. If there are suitable machines available, then the Matchmaker service will choose one and send the address of this machine and the identification string back to the client. The input of this request is: the job ID, the required CPU speed, the required memory size, the required disk size, the priority of the job. The response of this request is the address and identity of the chosen machine, if there is one available; otherwise, the request returns a null string.

This Matchmaker service is non-deterministic for two reasons. First, if there are several machines that satisfy a *jobSubmit* request, then the machine that is allocated can be nondeterministically chosen. Second, the Matchmaker service is implemented by two threads: one enqueues requests and one executes enqueued requests. Requests are enqueued in priority order, and is FIFO within each priority. Two servers S_1 and S_2 could behave differently because of these rules on priority. Consider two *jobSubmit* requests: r_h is a high priority request and r_ℓ is a low priority request. Let r_ℓ arrive at the servers before r_h . If server S_1 is slower than S_2 , then r_ℓ may arrive at S_1 when it is busy and arrive at S_2 when it is not busy. If r_h arrives shortly thereafter, then S_1 will execute r_h before r_ℓ and S_2 will execute r_ℓ before r_h .

3.2 State transfer

To fully understand the sources of overhead in state transfer, we compared the implementation using OGSi notifications for state updates (labeled *Notification*) to two alternative implementations, one that uses direct Grid service method calls (labeled *Call*) and one that uses TCP connections (labeled *Socket*). In the following tables we present the median, the mean, and the standard deviation for a set of 20 round-trip measurements. To better understand the overhead of state updates, our service can be configured to send an arbitrary amount of data in each state update.

First, Table 1 shows the round-trip time of a single state update, for a number of different state sizes (from 10 bytes to 100 kilobytes), as measured on the primary. Not surprisingly, *Socket* always has the smallest round-trip time, but this advantage goes from around 200 times faster for 10B updates to only 1.4 times faster when the state size is 100 kB. *Call* has intermediate round-trip times: at 10B, it is about 20 times slower than *Socket*, while at 10 kB it is only 2.5 times slower.

Note that the round-trip times for *Notification* are mostly insensitive to the size of the state update. Essentially, the cost of sending 10 bytes and 100 kilobytes with a notification is roughly the same. We think the cause of this lies in the format of GT3 notification messages; this is something that might be worth examining for later versions of the toolkit.

We also observed very high variance in samples: the standard deviation is sometimes higher than 50% and in one case is larger than the mean. This last case is due to a single outlier in the *Call* experiment for 100 kB of state update, which took 1.7 seconds. We think that much of this large variance is an artifact of Java garbage collection or other background processing in the Java

Table 1. State transfer round-trip time (milliseconds)

		10 B	100 B	1 kB	10 kB	100 kB
Notification	Median	192.5	189.0	193.0	195.5	190.0
	Mean	201.3	191.8	196.7	199.8	192.7
	St. Dev.	29.8	13.0	15.5	23.6	14.8
Call	Median	19.5	19.0	26.5	30.0	209.0
	Mean	26.4	24.2	33.1	32.6	299.0
	St. Dev.	12.0	9.2	17.9	6.3	333.0
Socket	Median	1.0	2.0	2.5	12.0	133.0
	Mean	1.5	1.7	2.5	14.8	144.5
	St. Dev.	0.8	0.5	0.5	11.9	32.4

Table 2. Client request round-trip time (milliseconds)

		10 B	100 B	1 kB	10 kB	100 kB
Notification	Median	242.0	241.0	240.0	238.5	232.0
	Mean	252.3	247.4	251.2	301.1	241.2
	St. Dev.	41.9	36.5	36.6	257.9	34.2
Call	Median	65.0	72.0	76.5	74.0	261.0
	Mean	71.4	78.0	89.7	82.5	350.8
	St. Dev.	21.9	28.9	40.9	21.9	333.5
Socket	Median	45.5	47.0	48.5	55.5	182.0
	Mean	52.8	56.5	55.5	62.6	195.9
	St. Dev.	21.6	26.6	21.8	22.9	50.4

virtual machine or the Grid container.

Table 2 shows round-trip times of client requests during normal, failure-free operation, as measured by the client. We would expect these numbers to be, approximately, the sum of the request round-trip time without primary-backup replication plus the state update overhead shown in Table 1 above. That is, indeed, the case since the request round-trip time of a normal Grid service, without replication, is 54.3 ms on average with the median being 44 ms. The data from the previous two tables is summarized graphically in Figure 5, where client request round-trip time is broken down into interaction between the client and the primary (white) and the interaction between the primary and the backup (solid, upward diagonal, and downward diagonal).

In Table 3, we normalized the data of Table 2 by dividing the median and mean numbers by the median and mean of the normal Grid service round-trip. So, each number shows the magnitude of overhead imposed by replication. The table shows that with *Socket* the median overhead of replication is small: for small state sizes (up to 10 kB) is 30% or less. With *Call*, the median overhead is 70% or less for small state sizes. For large state sizes all approaches perform similarly, with

overheads of 400% and more.

From these results, we conclude that notifications are considerably less efficient than socket messages and service calls for small state sizes. For larger state sizes all of the three approaches impose a high overhead. Note that in all cases the requests have very low overheads. In this situation a request that used to take 44 ms ends up taking between 4 and 6.5 times as long with replication. For Grid services that have longer-running requests, the overhead of replication will be diminished. For example, for a request that takes 3 seconds to execute and has state size of 100 kB, the overhead of replication is less than 10%. Hence, the drawback of using GT3 to implement primary-backup becomes negligible for long-running requests.

3.3 Failover

Another important metric for the performance of a fault-tolerant system is failover duration. This is a sum of two quantities: the time it takes for the backup to detect the failure, and the time it takes for the backup to notify the clients of a failover. The first quantity depends on the frequency of heartbeat messages and is

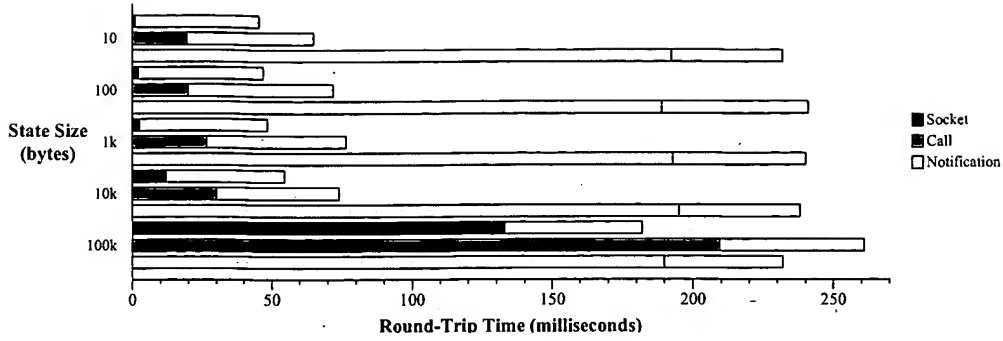


Figure 5. Median Round-trip times for state transfer using *Socket*, *Call* and *Notification* with different state sizes. The white portion is the round-trip time of a client request without replication. Therefore, the overall size of the each bar is the total client round-trip time.

Table 3. Ratio of client request round-trip with primary-backup to median/mean client round-trip without replication

		10 B	100 B	1 kB	10 kB	100 kB
Notification	Median	5.5	5.5	5.5	5.4	5.3
	Mean	4.6	4.6	4.6	5.5	4.4
Call	Median	1.5	1.6	1.7	1.7	5.9
	Mean	1.3	1.4	1.7	1.5	6.5
Socket	Median	1.0	1.1	1.1	1.3	4.1
	Mean	1.0	1.0	1.0	1.2	3.6

Table 4. Failover duration (milliseconds)

	1	2	4	8	16
Median	189	215	524	896	1989
Mean	194	302	547	1018	2269
St. Dev.	21	352	83	454	666

largely independent of the implementation. Therefore, we only measure the second quantity, as shown in Table 4.

With one client, it takes 194 ms on average to notify the client of a failure. As the number of clients increases, the notification overhead increases linearly. In [17] we report that recovering a network connection endpoint in less than 200 ms requires significant investment in equipment for logging of packets that may be lost due to the failure, and so we believe that 194 ms is quite acceptable. If the number of clients that shares the same instance is large, however, then the overhead may become too large. Again, these results were obtained

based the current implementation of GT3. A later version may be able to have notifications run faster than linear in the number of sinks.

Note that if the client doesn't have outstanding requests to the primary service when the failure happens, then the overhead of the failover at client is almost zero, since the client only needs to change the address of the service invoked.

4 Conclusion

Fault tolerance of stateful Grid services is becoming increasingly important with the development and use of OGSi. Both the infrastructure services such as monitoring, resource allocation, and scheduling, as well as Grid applications implemented as Grid services, are required to be reliable and highly available. In this paper, we showed that the facilities defined in OGSi and the newly proposed WS-Notification extension to Web services [11] can be used to design a primary-backup service. While not described in this paper, this service can be easily extended to multiple backups and

plified version of the Condor Matchmaker service [14]. Nondeterminism arises in this service both from the way resources are selected and from priorities.

Section 3 gives the performance results. We found the performance penalty was, in fact, quite high. While some of this may result from the lack of performance tuning in GT3, we believe that our findings also have larger implications related to how and where replication should be used to provide fault tolerance in Grid service architectures.

We do not consider client failure in this paper. One of the attractions of the primary-backup approach is that it defines a very simple client-server protocol that does not depend on clients being reliable. In other words, the correctness of the server, in terms of how it responds to requests, does not depend on help from the clients, which means that client failures can be dealt with using orthogonal approaches such as timeouts and leases [15]. We also do not consider software bugs that can lead to completely correlated failures. In this case, the primary and all backups could simultaneously crash. Again, there are separate techniques that are used in practice for tolerating such failures.

2 Architecture

We first describe the primary-backup approach to replication. We then cover the concepts behind Grid services, and then give a design of a primary-backup service on top of OGSi.

2.1 Primary-Backup replication

Primary-backup is a well-known technique for making services highly available [1, 5, 6]. A client sends a request to the primary, which receives and executes the request. The primary then sends a state update message to the backups and replies to the client. Typically, the primary does not reply to the client until it knows that all backups have received the state update. This is done to ensure that the backups are always consistent with the client: it is impossible for the client to know that the primary executed the request without the backups also knowing this. Figure 1 shows a space-time diagram of the execution of a simple primary-backup protocol.

Primary-backup requires that a client be notified that the primary has failed and allow the client to rebind to the newly-appointed primary. Ideally, this ability should be available below the level of the service request: doing so allows a client designed to interact with a single non-replicated server to be transparently ported to interact with a primary-backup service.

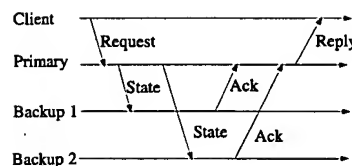


Figure 1. Primary-Backup Protocol

Primary-backup has a few drawbacks. First, it is best suited for tolerating benign failures such as crashes and message loss, rather than arbitrary or malicious failures. Unless malicious failures are a concern in a specific Grid environment, we consider masking only benign failures a worthwhile tradeoff for the ability to run non-deterministic applications. Another drawback is that primary-backup requires that the environment is synchronous enough to support the use of heartbeats to detect failures. In practice, this means that the primary and the backups need to run on a cluster that is managed by some failure detector and recovery manager. Commercial products, such as the VERITAS Cluster Server [16], can be used for this purpose.

2.2 Grid Services

Grid services in OGSi combine Grid technologies with Web services to provide a platform for building distributed applications. Unlike Web services, Grid services are stateful and may be short-lived. The OGSi model allows each client to choose among several available instances of a service or create its own instance. The instances may have a limited lifetime since resources can be created to serve certain clients and are removed after they are no longer needed.

Interaction between a Grid service and a client happens in a request-reply fashion using strictly-defined interfaces and a certain encoding of data (interfaces are described by WSDL and the messages are encoded using SOAP, both of which are XML-based). In addition to regular requests and replies, Grid instances may subscribe using an OGSi-specified interface to *notifications*, which asynchronously alert subscribers (called *sinks* in OGSi terminology) of state changes. Using notifications avoids wasteful polling.

Notifications can be of two types: a *push* notification sends information along with the notification, whereas a *pull* notification is used to indicate that something has changed: it is up to the notification subscriber to request (or pull) the information using a regular request. Pull notification gives the subscriber the freedom to decide whether and when to get information associ-

ated with the notification, while push notification avoids the overhead of that additional call in situations where the information is needed immediately.

2.3 Primary-backup for Grid Services

There are three general problems that any implementation of a primary-backup mechanism needs to solve:

1. *Transfer of application state.* Before replying to the client, the primary needs to send the change in its state to the backups. A reply can be sent to the client only when it is known that the backups will eventually apply the state change.
2. *Detecting failures.* Crashes and lost messages need to be detected. This is normally done by setting a timeout for every message. If no messages are sent for a long time, then a *heartbeat* message can be sent to check on a machine.
3. *Switching to a new primary.* Originally, one of the service instances is designated as a primary and others as backups. After a failure of the primary, the backups agree on a new primary and ensure that all future requests are directed to it.

Grid service notifications are a natural mechanism for solving all three of these problems because state updates and failures are inherently asynchronous events. Also, notifications provide a simple mechanism for disseminating information to a number of interested parties—several backups may be interested in the same state update, and several clients may be interested in the same failure notification. Consequently, in our system, backups register with the primary as sinks for state update notifications and heartbeat notifications, and each client registers with each backup as a sink for the failover notification that tells it to switch to a different primary and to resend the last request if it was expecting a reply. We use a push notification for the state transfer because the backup needs every state update. For heartbeat and failover, pull notifications are used because there is no data associated with those events.

The normal execution proceeds as follows. A client makes a Grid service request to the primary, which executes the request. When execution ends, the change in the state of the service is extracted and sent to the backups via a notification. When the primary collects acknowledgments from all backups it replies to the client. The state extraction and injection are application-specific: the Grid service needs to support

methods that allow this to be done. In addition, the service can be designed to have the primary send checkpoints to the backups if its computation is long-running.

Failure of the primary is detected by backups when they do not receive a heartbeat message after a certain period of time. This method allows detection of host and task crashes, as well as network partitions. At that point, the backups need to cooperate in election of the new primary. The newly elected primary then sends a failover notification to the client so it can obtain a new server instance handle. If the client was expecting a reply from the service when a failover notification arrives, then the client resubmits the request to the new service instance. If the old primary had already sent a state update to the new primary, then the new primary can reply with the result computed by the old primary. Otherwise, it can compute the result itself (perhaps starting from a checkpoint if the primary had sent checkpoints to the backups).

Failures of the backups do not interfere with the operation of the surviving system components, so the only new issues are the detection of backup failures and the integration of new backups into the system. Neither is conceptually hard to implement, although integration of a new backup may require a large amount of state to be transferred. The details of how to best do this are outside the scope of this paper.

2.4 Implementation details

In this section we give a more detailed overview of our system using pseudocode to illustrate key actions performed by each of the three participants: a client, a primary, and a backup. Each one is enclosed in an object with private variables and methods. Note that we use C language convention for pointers: $\&x$ is a reference to variable x .

The client code, shown in Figure 2, is interposed between the client application and the original SOAP stub in such a way that client code is not changed. The original stub supports the `INIT` method, which is called when the client binds to a Grid service, and a number of operations, shown here collectively as `OP`. We intercept `INIT` with the `INIT_CLIENT` method to register for receipt of failover notifications from each backup. The `OP` method spawns a separate thread, implemented by `INVOKE_OP`, to invoke the operation via the original stub.

If the primary crashes during this invocation, two things will happen in arbitrary order: the call to `stub.op()` will return an error message, and a failure notification will arrive from the backup, causing `FAILURE_HANDLER` method to execute. To reduce the

to dynamically adding backups. In addition, by using slightly modified client stubs, failover can be done transparently to clients. We did need to make strong assumptions on failure detection, but they can be satisfied by existing commercial software.

We found the overhead of using GT3 implementation of the OGSI notification to be quite high. The overhead is particularly large in the cases where the state data is small or the number of clients is large. Much of the overhead seems to come from the cost of notifications, which can most likely be improved in future implementations of GT3. Failing that, one might wish to provide state update below the OGSI level or by using simpler OGSI facilities such as basic Grid service method calls. It might be possible to improve performance of primary-backup by using an alternate protocol binding—something that is specified in OGSI but not available in GT3—but we have not explored this option in any detail.

Our approach for primary-backup is only applicable for replicas located in a cluster, since otherwise failure detection becomes too unreliable for primary-backup. We are currently looking at methods that still accommodate nondeterminism, like primary-backup, but that can work in a wide-area network where asynchronism is more of an issue.

References

- [1] P. Alsberg and J. Day. A principle for resilient sharing of distributed resources. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 562–570, Oct 1976.
- [2] Y. Amir, B. Awerbuch, and R. S. Borgstrom. Managing checkpoints for parallel programs. In *Proceedings of the 1st International Conference on Information and Computation Economics (ICE-98)*, 1998.
- [3] A. Amoroso, K. Marzullo, and A. Ricciardi. Wide-area Nile: a case study of a wide-area data-parallel application. In *Proc. 18th International Conference on Distributed Computing Systems*, pages 506–515, Amsterdam, The Netherlands, May 1998.
- [4] A. Beguelin, E. Seligman, and P. Stephan. Application level fault tolerance in heterogeneous networks of workstations. *Parallel and Distributed Computing on Workstation Clusters and Networked-based Computing*, Jun 1997.
- [5] K. Birman, T. Joseph, T. Rauechle, and A. Abbadi. Implementing fault-tolerant distributed objects. *IEEE Transactions on Software Engineering*, SE-11(6):502–508, Jun 1985.
- [6] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg. Primary-backup protocols: Lower bounds and optimal implementations. In *Proc. 3rd IFIP Conf. on Dependable Computing for Critical Applications*, pages 187–198, Mondello, Italy, September 1992. Springer-Verlag, Wien.
- [7] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, 1999.
- [8] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.
- [9] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.
- [10] Global Grid Forum. *Final OGSI Specification V1.0, Proposed Recommendation*, Jul 2003.
- [11] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratham, J. Parikh, S. Patil, S. Samdarshi, S. Tuecke, W. Vambenepe, and B. Wehl. Web service notification (ws-notification), Jan 2004. <http://www.ibm.com/developerworks/library/ws-resource/ws-notification.pdf>.
- [12] S. Hwang and C. Kesselman. Grid workflow: A flexible failure handling framework for the grid. In *Proc. 13th IEEE Intl. Symp. on High Performance Distributed Computing (HPDC-13)*, pages 126–137, Seattle, Washington, USA, June 2003.
- [13] M. Livny and J. Pruyne. Managing checkpoints for parallel programs. In *Proceedings of IPPS Second Workshop on Job Scheduling Strategies for Parallel Processing*, 1996.
- [14] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC-7)*, 1998.
- [15] F.B. Schneider. Implementing fault-tolerant services using the state-machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.
- [16] Veritas company homepage. <http://www.veritas.com/index.html>.
- [17] D. Zagorodnov, K. Marzullo, L. Alvisi, and T.C. Bresnoud. Engineering fault-tolerant TCP/IP servers using FT-TCP. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 393–402, San Francisco, California, USA, June 2003.



AberdeenGroup

Sun's Grid Computing
Solutions Outdistance
the Competition

An Executive White Paper

May 2002

Aberdeen Group, Inc.
One Boston Place
Boston, Massachusetts 02108 USA
Telephone: 617 723 7890
Fax: 617 723 7897
www.aberdeen.com

Sun's Grid Computing Solutions Outdistance the Competition

Executive Summary

Sun Grid Engine (SGE), SGE Enterprise Edition (SGEEE), and Platform Computing's Load Sharing Facility (LSF) are three distributed resource management software solutions. The source codes for SGE and SGEEE are available in the Grid Engine open source project sponsored by Sun Microsystems and hosted by ColabNet. The Grid Engine open source project was launched in June 2001. Reference releases are available via downloads from the Grid Engine open source project (www.gridengine.sunsource.net).

The downloads are 100% compatible with the Sun products, SGE and SGEEE, apart from a different binary code license. SGE is free and available only through downloads from www.sun.com/gridware. SGEEE is available from Sun and its resellers only on CD at a cost that is dependent on grid size.

SGE and LSF provide comparable functionality and are suitable for what Sun refers to as cluster grids (one-project, one-department grids). SGEEE provides distributed resource management in more sophisticated grids — i.e., enterprise grids (multiple projects or departments, one organization) — and provides the basic functionality for global grids (i.e., collections of enterprise grids that cross organization boundaries). SGE also provides basic functionality for global grids, but not as much as SGEEE.

Moving from single departmental grids to more sophisticated grids requires human cooperation. SGEEE implements the concept of policies for specifying how humans will cooperate in multidepartment, multiproject grids. That is, policies define how computer resources are distributed among projects and people. For example, some users may not want to share their CPUs (central processing units) on certain days of the week or with specific groups. As a result, policies are implemented at a level above the distributed resource management layer.

SGEEE's policy module provides the following four benefits, which neither SGE nor LSF provide:

1. It introduces new utility computing-like parameters for scheduling. Scheduling jobs is not based on priority alone. With SGEEE, a user, team, department, or project can receive a resource allocation, for a period of time, based on some percent of the total resources available. SGEEE will ensure that the assigned percentage of resources is available to the jobs within that project or for a user, team, or department.
2. It enables collaboration. Users and project teams can negotiate resource assignments that can vary from week to week. For example, a project team may get 10% of the resources this week and 30% of the resources next week. This type of negotiation allows project teams to better manage the start and completion of their projects and allows a project team

with a "hot" project to negotiate sufficient resources to ensure that the project can finish on time.

3. It offers management alignment of compute power (cumulative usage over time) to specific projects that have high importance.
4. The more you pay the more you get. Cumulative resource usage can be proportional to budget contributions.

Sun and Platform Computing have taken different approaches to developing grid computing solutions. Sun delivers grid computing solutions with necessary functionality integrated within a single product such as SGE/EE; whereas, Platform's approach is to provide a base product (LSF Base combined with LSF Batch) and then surround it with other products to complement the functionality in the base offering. Even with the purchase of additional Platform products such as LSF MultiCluster and LSF Parallel (beyond LSF Base and LSF Batch), LSF cannot provide the functionality available in SGE/EE.

In this *Executive White Paper*, Aberdeen analyzes SGE, SGE/EE, and LSF and draws the following conclusions:

- SGE and LSF provide comparable functionality and provide basic resource management for cluster grids.
- SGE/EE is the most comprehensive, cost-effective grid computing solution currently on the market, one that is capable of providing resource management for enterprise grids and aspects of global grids.
- Sun's most serious competitors in grid computing — HP (Hewlett-Packard) and IBM — are relying heavily on third-party proprietary products from companies such as Platform Computing versus building their own. It is Aberdeen's perspective that Sun, by leveraging the contributions of the Grid Engine open source project, is able to provide superior functionality at a lower cost.
- While Sun is ahead of its competition, building its own grid products, it is incorporating concepts developed by the leading research organizations (of which Sun is a primary participant) in grid computing — the open source Globus project (www.globus.org), the Global Grid Forum (GGF), and the Distributed Resource Management Application API (DRMAA) working group (which Sun initiated with Intel and Veridian) within GGF.

What Is Grid Computing?

Today, computing is often concerned with collaboration, sharing of data files and databases, and other means of interaction across departments within an organization and across organizations. These requirements have led to new ways of performing application development and deployment. The requirements have been,

for the past few years, primarily the concern of developers of distributed systems for scientific and technical research. Work within this community has led to the formation of grid computing technologies.

In general, the specific problem that grid computing tries to solve is coordinated resource sharing in dynamic, multi-institutional virtual organizations (VOs). Examples of VOs include service providers, manufacturers, and organizations involved in collaborative problem solving. This description of the problem that grid computing tries to solve aligns perfectly with Sun's notion of global grids, which span organizations. However, cluster grids and enterprise grids are also real examples of grid computing. At Sun, grid computing is defined as the pooling of resources into virtual systems. Sun introduces scaling with cluster grids, enterprise grids, and global grids.

Sharing is concerned with direct access to computers, software, data, and other resources, i.e., the type of sharing required for collaborative problem solving. Sharing is highly controlled, with resource providers and consumers defining what is shared, who is allowed to share, and the conditions under which sharing occurs. Sharing relationships must be flexible for sophisticated and precise levels of control of how shared resources are used. On the other hand, the relationships must be definable so that they can be implemented and enforced.

Resource sharing policies are conditional statements. That is, a resource owner makes resources available subject to constraints on when they can be used, how they can be used, and for what they can be used. The implementation of constraints requires mechanisms for expressing policies, for establishing the identity of a consumer or resource, and for determining whether or not the use of a resource is consistent with the specified sharing relationships. Any policy mechanism must be capable of handling relationships that can vary dynamically over time, in terms of the resources involved, the nature of the accesses permitted, and the participants to whom access is permitted. In addition, a new participant (individual, group, or organization) must be able to "discover" the nature of relationships that exist at any given time. For instance, a new participant must be able to determine what resources are available for it to access, the quality of the resources, and the policies that govern access. A single resource may be shared in several ways, possibly in different ways for each participant.

Today, distributed computing technologies do not address many of the concerns listed for sharing resources. The Internet, for example, addresses information interchange among computers, but it does not address the types of flexible policies required for sophisticated resource sharing at various computer sites by individuals, groups, and organizations. That is where grid computing becomes important. In the past few years, researchers within the grid community have produced protocols, services, and tools that address the challenges of resource sharing. These

technologies include security solutions, resource management protocols, and policies that support secure remote access to computing and data resources, etc.

Members of the Globus Project (www.globus.org) have published an Open Grid Services Architecture (OGSA) proposal. The OGSA proposal outlines interfaces to grid computing software that comply with Web services standards. The objective is to take advantage of Web services properties such as service description and discovery. If this proposal is adopted, common grid services like job scheduling, authentication, failure detection, and migration of data will all be accessed through standard Web services architecture. Web services are a natural fit with the underlying services for grid computing because Web services aim to connect applications across large heterogeneous networks using Web-related standards like Web Services Definition Language (WSDL) and eXtensible Markup Language/Simple Object Access Protocol (XML/SOAP).

While grid computing originated in the research world and few, if any, commercial grids are in use today, it is Aberdeen's perspective that grid computing will also become an important way to deploy commercial applications including e-Business applications. The jump to the commercial world is not difficult because groups work in teams and share resources just like scientific researchers do, and they want the same three things:

1. Efficient use of hardware and workload balancing;
2. Quality of service — which nodes are working, which nodes are not working, route around overloaded nodes, etc.; and
3. Flexibility — virtual access to resources.

Grids are a potential solution to an organization's lack of compute power and to a more efficient use of existing resources. In many companies, desktop utilization is relatively low, on the order of 20% or less, and there is often a duplication of resources across groups. Grids, via distributed resource management software, aggregate available compute resources and deliver compute power as a network service. Grids increase productivity by matching workload and resources. And, perhaps more importantly, grids make it as easy to use many CPUs as to use one, allowing the user to be much more productive.

Software Requirements for Grid Computing

Grid computing requires a collection of software features that contribute to managing the resources in heterogeneous, distributed computing environments. It is Aberdeen's perspective that sophisticated grid computing solutions must address the following:

- *Coordinated resource management* — managing the use of shared resources to best achieve an enterprise's goals such as productivity, timeliness, level of service, etc.
- *Dynamic policy mechanisms* — mechanisms capable of handling relationships that can vary dynamically over time, in terms of the resources involved.
- *Discovery* — participants must be able to determine what resources are available for access, the quality of resources, etc.
- *Dynamic scheduling* — the capability to remove resources from a low priority job and give them to a higher priority job so the higher priority job can complete execution. This is done without killing the lower priority job. If no other higher priority jobs are remaining, then the resources can be given to lower priority jobs to continue execution.
- *Controlled sharing of resources* — enforce policies that constrain access according to group membership, ability to pay, etc.
- *High-level policy administration* — distribution of computer resources among projects and people versus the lower level concept of distributing resources among jobs.
- *Security* — security services define standard functions for identifying individuals in communicating parties, encrypting messages, and so forth.
- *Heterogeneous, distributed computing environments* — grids that span projects and organizations almost always utilize platforms from several suppliers.
- *Checkpointing capability* — ability to move jobs from host to host during execution without restarting.
- *Open standards* — standards-based solutions facilitate extensibility, interoperability, portability, and code sharing.
- *Adjust to various customer attitudes toward grid computing* — architectures should be flexible enough to satisfy distributed applications and satisfy user application requirements versus the grid architecture's forcing a structure on the applications.
- *Differences with respect to high-performance computing (HPC)* — grid computing and high-performance computing are intertwined, but they are not the same. Some HPC applications may suffer using some grid computing approaches, e.g., bandwidth- and latency-dependent applications deployed across multicluster grids or grids spanning distances.

Market Potential for Grids

Grids have the potential to impact the marketplace in a number of ways by enabling more complex simulations and modeling efforts, expanding and sharing access to databases, and speeding communication between collaborators working on common projects.

Today, grids are in the very early adopter stage with the primary use coming from research, engineering, science, and areas such as Electronic Design Automation (EDA), life sciences, and others. The primary drivers are resource optimization, access to resources, cost sharing, and improved models for managing resources.

Sun's SGE and SGEEE

Sun has two grid computing solutions — SGE 5.3 and SGEEE 5.3. SGE is suitable for cluster grids, and SGEEE is capable of handling enterprise grids and some aspects of global grids. When Sun announced SGE in September 2000, the product already had a five-year history. Sun acquired Gridware, developer of the initial product, in July 2000.

The front-end development for both SGE and SGEEE is done in the Grid Engine open source project (www.gridengine.sunsource.net). Sun does not deviate from the source code produced via the Grid Engine project for releases of SGE/EE. Reference releases, which are functionally identical to SGE and SGEEE at a point in time, are available via the Grid Engine project. SGE and SGEEE are both made from the same source tree in the Grid Engine project and share internal components. When Sun decides to release a new version of SGE and SGEEE, it brings a stable build of Grid Engine software into the Sun quality assurance process and documents and productizes the software under the SGE/EE brands.

For those users who download SGE from www.sun.com/gridware and who buy SGEEE, Sun delivers controlled upgrades via patches (through support services contracts). Sun does not provide support contracts on the binaries available in the Grid Engine open source project. But most enterprise customers want a supported version and are willing to pay for support. Anyone — including Sun's competitors — can use the Grid Engine open source project code to make their own version of Grid Engine.

Product Comparisons

Functionality and feature comparisons for SGE, SGEEE, and LSF are presented in the following sections, and Appendix A contains detailed overviews of the three grid product offerings.

Comparison of SGE 5.3 and LSF 4.2

Both SGE and LSF provide functionality for what Aberdeen considers to be basic resource management — both solutions are suitable for supporting cluster grids,

but not enterprise or global grids. One of the major differences between SGE and LSF is that SGE is free and LSF Base plus LSF Batch costs \$995 per CPU for Unix and \$399 per CPU for Linux. Support for SGE is available from Sun via the Web and via a software-only support contract with Sun Enterprise Services. Support is also available from Sun partners and resellers.

Another difference between SGE and LSF is that SGE is a solution with all the functionality required to provide resource management for cluster grids integrated within a single product offering. Platform Computing's approach is to provide a base product (LSF Base with LSF Batch) and surround it with other offerings — LSF MultiCluster, LSF Parallel, LSF JobScheduler, LSF Make, and LSF Analyzer — to complement the functionality in the base offering.

LSF is built in layers. For this reason, LSF is composed of several individual products. The base system (LSF Base) provides a basic level of services that allow users to perform dynamic load sharing and distributed processing. However, if sophisticated job scheduling and resource allocation policies are necessary — as they frequently are in grid computing — more complex scheduling must be built on top of LSF Base using LSF Batch. (LSF Batch is a batch system built on top of LSF Base to provide distributed batch job scheduling services.) That is the reason LSF Base and LSF Batch are both necessary to provide a base product for grid computing. The disadvantage of Platform's approach is that the user has to pay additional costs for each of the separate offerings. A description of the additional LSF products can be found in Appendix A.

SGE and LSF both provide the following functionality:

- *Batch processing* — submit jobs and process them as soon as the resources are available;
- *Dynamic allocation of resources* — allocate resources as they are required and release them when not needed;
- *Fault tolerance* — automatically resubmit jobs that fail;
- *Failover capability* — grid continues to operate if one or more hosts fail;
- *User-specifiable resources* — at submission time, a user can specify resources needed to complete a job;
- *Resource location independence* — the user does not know or care where the compute resources are located in the grid;
- *Job status* — users want to know what is happening to their job at any given time;
- *Host status* — system administrators need to know the utilization and up/down status of all hosts in a grid;

- *Centralized management* — system administrators need the capability to manage an entire grid from one application, which is not the same as having a single machine from which to manage a grid; and
- *Suspend/resume jobs* — the user (and system administrator) has the capability to halt a job and restart it later without losing the work already completed.

Table 1 provides a comparison matrix of important features for SGE 5.3 and LSF 4.2.

The concept of queues in SGE and LSF requires some explanation because they affect scheduling in the two product offerings. When a job enters an LSF queue, it must remain in the queue until its execution is completed, unless a system administrator manually moves it to another queue. LSF queues are distinct from hosts. All LSF jobs in a high-priority queue run before any jobs in a lower priority queue are started.

SGE queues are an expression of what resources are available on each machine in a grid. When a job is submitted to an SGE-based system, the SGE scheduler takes into account the order in which the job was submitted, what queues (hosts) are available, and the priority of the job. The scheduler places all jobs in a single pending list and continuously re-evaluates the availability of resources and priority of all jobs in the grid. If a host has the resources available, SGE will automatically move the highest priority job to a queue on that host to begin execution immediately. That is an advantage for SGE (and SGEEE) with respect to utilization of resources and throughput.

An analogy for LSF queues is a grocery store where customers select a queue and wait in line to be checked out. An analogy for SGE queues is a hospital emergency room where selection of those served next is made on a number of parameters, with the list frequently re-sorted.

SGE Versus SGEEE

SGEEE contains all of the functionality and features that SGE has plus the important concept of policies that permit it to adapt to general grid computing models such as enterprise and global grids. SGEEE is substantially different than SGE (and LSF or other similar competitive products). The primary difference is SGEEE's policy module.

Any SGE grid can be upgraded to SGEEE by upgrading the master host in the grid. SGEEE requires that the master daemon run on a Solaris (2.6 to 9.0)/SPARC master host; whereas, the master daemon in an SGE grid can run on Solaris (2.6 to 9.0)/SPARC, Solaris (2.6 to 8.0)/x86, Linux/SPARC, or Linux/Intel. Master daemons in both SGE and SGEEE are 100% compatible to execute daemons from Grid Engine open source project builds such as AIX, HP-UX, IRIX, Linux, and others.

SGEEE Versus LSF

Aberdeen does not view LSF as a competitor to SGEEE because LSF does not provide the policies required to manage resources in enterprise grids. LSF is priced at \$995 per CPU for Unix. For more than 100 licenses, SGEEE is priced at approximately \$300 per CPU. Not only is SGEEE much less expensive than LSF, but it also contains significant functionality that LSF does not provide.

SGEEE Versus Aberdeen's Software Requirements for Grid Computing

When the functionality and features that SGEEE provide are compared with the list of software requirements that Aberdeen considers important for grid computing, SGEEE fares very well. For instance, it is Aberdeen's perspective that SGEEE is more than adequate for controlling and resolving resource sharing in dynamic VOs where collaborative problem solving is a must. SGEEE's policies are capable of defining what is shared, who can share, etc. And, importantly, policies defined by the use of SGEEE's policy mechanism can be implemented and controlled.

SGEEE provides for dynamic scheduling and controlled sharing of resources. It thrives in heterogeneous, distributed computing environments; is based on open standards; provides an adequate degree of security across departments and organizations; and is flexible enough to allow jobs to be automatically re-directed to new hosts when resources become available.

Table 1: Comparison of Important Features — SGE Versus LSF

Features	SGE 5.3	LSF 4.2
Cost	Free	\$995 per CPU for LSF Base and LSF Batch for Unix; \$399 per CPU for Linux
Dynamic scheduling	Yes	No
Transparent to application	Transparent to application if compatible with the underlying operating system	Transparent to application if compatible with the underlying operating system
Scripts for job submission	Scripts specific to the environment are required	Scripts specific to the environment are required
Ease of installation	Very easy to install, takes one to two minutes per host	Takes about a half-day to create a grid of 50 to 100 hosts
Ease-of-use	Easy to use (based on information collected at download time)	More difficult to use than SGE (based on download information)
Scalability	There are separate daemons for the master and scheduling functions that can run on separate CPUs in a dual processor host, allowing more jobs to be processed simultaneously	Places master and scheduler functions in one daemon, creating potential delays in scheduling

Features	SGE 5.3	LSF 4.2
Enterprise grid-like capability	Not available in SGE, but available in SGEEE	Limited form using LSF MultiCluster at an additional cost of \$200 to \$300 per node
Specify queue name at job submission	Not applicable	Name for a specific queue can be submitted with the job
Array memory for submitting large numbers of jobs at the same time	Yes	Yes
Globus support	Supported under SGE/EE	Yes
Parallel job execution	Yes	With LSF Parallel at an additional cost
Heterogeneous computing environments	Supports AIX, HP-UX, IRIX, Solaris, Tru64 UNIX, Linux	Supports AIX, HP-UX, Solaris, Tru64 UNIX, Linux, Mac OS X, Windows
Master host configurations	Each grid requires a master host. SGE master daemon must run on Solaris/SPARC, Solaris/x86, Linux/SPARC, or Linux/Intel. SGEEE master daemon must run on Solaris/SPARC	Each grid requires a master host. No platform restrictions on master host.
Job checkpointing	Yes	Yes
GUI support	Yes	Yes
Command line interface	Yes	Yes
Error logging	Yes	Yes
Required resources can be requested at job submit time	Yes	Yes
Job accounting, e.g., submit, execution times	Yes	Yes
Job arrays	Yes	Yes
SNMP agent support	No	Yes
Adding/removing hosts without shutdown	Yes	Yes

Source: Aberdeen Group, May 2002

Aberdeen Conclusions

While grid computing originated within the scientific and technical market segments, it is just as appropriate for commercial applications and any type of computing where sharing of files and databases as well as collaborative forms of interaction across projects, department, and organizations are important. With SGE 5.3 and SGEEE 5.3, Sun is able to provide the functionality required for today's grids. And, as grid computing moves more and more to the forefront, Sun, with SGEEE 5.3 and its concept of policies, is positioned to be the leading supplier of grid computing products for the future.

Sun's most serious competitors in the scientific and technical market segments where grid computing originated — HP and IBM — are trailing Sun in the development of grid product offerings. While Sun has jumped out front in the grid market with its own products — SGE 5.3 and SGEE 5.3 — the competition is relying on third-party offerings such as Platform Computing's LSF. Aberdeen concludes that LSF 4.2 falls far short of SGEE 5.3 in the functionality required to create sophisticated grids, and LSF 4.2 costs significantly more per CPU than does SGEE 5.3.

Sun is positioned to respond to new grid computing requirements through its customer base and through its involvement with the open source Globus project (www.globus.org) and the GGF. Sun was the first sponsor of GGF in 1999. Sun is convinced that standards and open source are fundamental for the success of grid computing and suppliers and users. Sun was the first systems supplier to place key grid technology — the Grid Engine product suite — into open source. The Globus-based OGSA architecture is based on the same Web services standards that Sun ONE (Open Net Environment) Web Services are based on. Sun is providing input as an active member of the OGSA working group based on its experience with the large Sun ONE installed base.

Sun also initiated the DRMAA working group within the GGF. This effort promises to give commercial application vendors a "write once" interface to utilize whatever DRMAA-compliant resource management system the customer has deployed, thus reducing deployment effort for end-users. A number of manufacturers of resource management systems are working to complete this API in 2002.

APPENDIX A

Product Descriptions for SGE 5.3, SGEEE 5.3, and LSF 4.2

SGE 5.3

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
www.sun.com
(650) 960-1300

SGE 5.3

In SGE/EE terminology, an administrator is a user who is allowed to fully control SGE/EE. An operator is a user with administration privileges who is not allowed to change queue configurations. An owner is a user who is allowed to suspend jobs in queues that he or she owns or disable owned queues. A user can manipulate only the jobs that he or she owns.

SGE Overview

SGE is layered above the operating system and requires no alterations to applications. SGE can be used with any type of server — dedicated or shared, compute farms, and desktop systems. SGE is suitable for developing cluster grids.

The basic function of SGE is to match available resources in a grid with users' requests. SGE supports both batch jobs and interactive jobs. A batch job is a shell script that can be executed without user intervention, and it does not require access to a terminal. An interactive job is a session started with SGE commands that open an X-terminal window for user interaction or provide the equivalent of a remote login session.

Computational resources are delivered to user jobs by SGE based on resource policies (not to be confused with the policies for people and projects available in SGEEE) specified by the grid cluster owner organization's technical and management staff. SGE uses the policies to examine available computational resources within the grid, gathers these resources, and then allocates them to jobs in a manner that optimizes their usage across the cluster grid.

SGE Job Flow

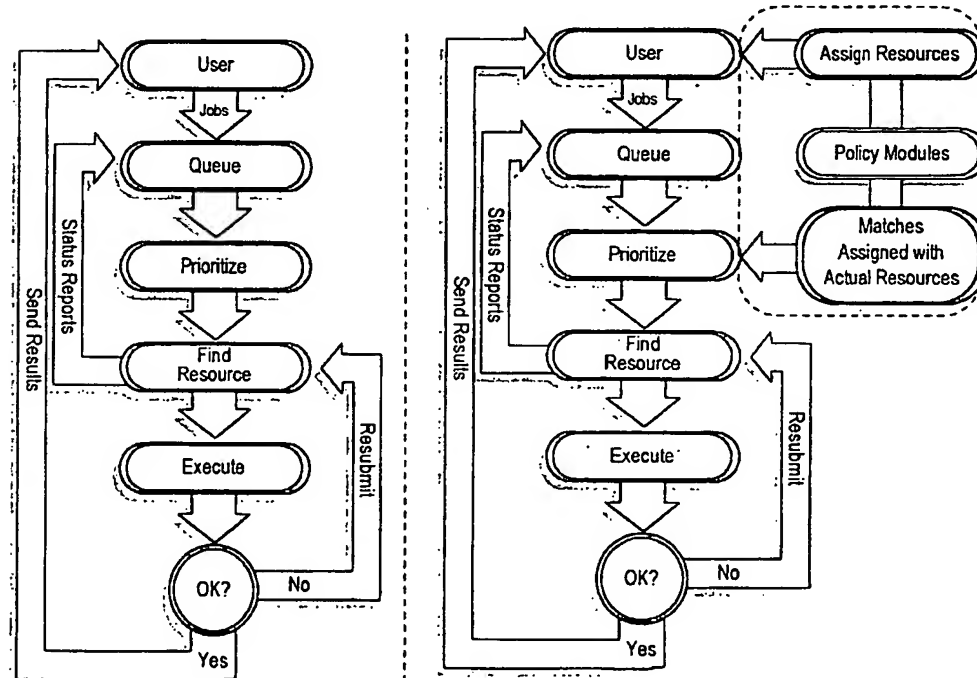
Users can submit batch jobs, interactive jobs, and parallel jobs to SGE. SGE supports checkpointing jobs — jobs that can migrate from host to host within a grid without user intervention and based on load demand on the SGE system. Checkpointing is a procedure that saves the execution status of a job into a checkpoint area for the job, permitting the job to be aborted and restarted later without loss of information and already completed work.

At a high level, SGE works in the following manner (Figure 1):

- Accepts jobs from users;
- Places jobs in a computer holding area until they can be executed;
- Sends them from the holding area to a host where they can be executed;
- Manages them during execution; and
- Logs a record of their execution when they are finished.

A user who submits a job to SGE specifies a requirement profile for the job along with user identification and a priority number. The requirements profile is a

Figure 1: Job flow for SGE (Left) and SGEEE (Right)



Source: Sun Microsystems, May 2002

statement of attributes associated with the job such as memory requirements, operating system required, etc. SGE schedules the most important jobs first. Based on the contents of the requirement profile, SGE dispatches the job to a suitable queue associated with an appropriate host server on which the job will be executed. SGE uses load balancing to spread workload among available servers.

SGE Components

SGE is composed of several components — hosts, daemons, queues, and client commands. An SGE grid cluster is composed of four types of host nodes — master, execution, administration, and submit. The master host controls all SGE activity. It runs the master daemon and the scheduler daemon. Implementing the master and scheduling functions in two separate daemons allows these two functions to run on different CPUs in the same host, thereby improving scalability. These two daemons control all queues and jobs and maintain tables about the status of queues and jobs, about user access permissions, etc. By default, the master host is also an administration host and submit host. Execution hosts are nodes that have permission to execute (and submit) SGE jobs, and they have queues associated with them.

An administrative host is a node from which administration commands may be issued. It is responsible for carrying out administrative activity for an SGE cluster grid. Submit hosts are nodes that are permitted to submit batch jobs and query their status. A node in an SGE cluster grid can belong to multiple host classes simultaneously.

The functionality of the SGE system is performed by four daemons. The master daemon maintains tables about hosts, queues, jobs, system load, and user permissions. The master daemon must run on a Linux- or Solaris-based host. The scheduler daemon maintains an up-to-date view of a grid's status. It determines which jobs are dispatched to which queues and then forwards its decisions to the master daemon, which initiates the required actions. The execution daemon is responsible for the queues associated with the host on which it runs, and it periodically forwards the status of jobs and the load on its host to the master daemon. The communication daemon communicates over a well-known TCP (Transmission Control Protocol) port. It is used for all communication among SGE components.

An SGE queue is a container for a class of jobs allowed to execute concurrently on a particular host. Throughout their lifetimes, running jobs are bound to a queue. SGE users do not submit jobs to queues. Users specify the requirement profile of a job, and SGE software dispatches the job to a suitable queue on the host with the lowest workload. The command line user interface is a set of commands that allows users to manage queues, submit and delete jobs, check job status, and suspend/enable queues and jobs.

SGEEE 5.3

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
www.sun.com
(650) 960-1300

SGEEE 5.3

The primary difference between SGE and SGEEE is that SGEEE can consolidate multiple cluster grids into enterprise grids. This capability is provided in SGEEE with the incorporation of policies. These SGEEE policies are a level above job resource allocation. Policies dictate how compute resources are distributed among projects and people, not jobs.

The administrator of an SGEEE grid defines customized policies corresponding to what is appropriate for the use of the grid. Policies are needed to provide the flexibility required for managing resources across multiple projects and across multiple organizations. Project owners using an enterprise grid have varying requirements with respect to how they manage their projects. They need capabilities to negotiate policies, flexibility for manual overrides for unique project requirements, and automatically monitoring and enforcement of policies.

Policies/Tickets

Within SGEEE, tickets are used to distribute the workload. They are assigned to projects, users, and jobs. More tickets mean higher priority and faster execution. The following are four policies available in SGEEE:

1. *Share-tree based*: This policy allocates a percentage of total compute resources to each user or project. If actual cumulative usage allocated over a period of time to each user or project exceeds the assigned value of resource allocation, then "borrowed" resources are "returned" to the other users. For example, if project A in a queue is not using all of its resources, then project B can gain access to them and use as much of A's resources as A allows. When A resumes normal activity, B must return to A a fraction of the borrowed resources. The longer B holds resources borrowed from A, the fewer resources B has to return to A. That is, if B holds some of A's resources for three weeks, then B has to return less than if B holds them for only two days.
2. *Functional*: The functional policy is similar to the share-tree-based policy, but B does not have to return resources "borrowed" from A. This policy forgets about the past.

3. *Deadline:* The deadline policy is a policy that is invoked whenever a job must be finished before or at a certain point in time and may require special treatment to achieve this. An administrator can manually give jobs extra resources to meet deadlines. The resources are relinquished after the deadline. The deadline policy is implemented by redistributing tickets — an administrator can assign a job more tickets to raise its priority so that it can meet its execution deadline.
4. *Override:* The override policy allows administrators to make resource allocation decisions manually instead of automatically by SGEEE. The override policy is implemented by giving additional tickets to jobs, users, or projects to temporarily adjust their relative importance.

SGEEE software uses these policies to examine the available computational resources within the enterprise grid. Then it gathers, allocates, and delivers these resources automatically so that highly optimized resource usage is achieved.

Figure 1 illustrates job flow for SGEEE. The job flows for SGE and SGEEE are similar in some respects; however, the availability of policies for resource management at the people and projects level imposes another level of control on top of resource management for jobs.

LSF 4.2

Platform Computing, Inc.
3760 14th Avenue
Markham, Ontario L3R 3T7
www.platform.com
(905) 948-8448

LSF 4.2

LSF is designed to be a layer on top of existing operating systems. LSF runs on most Unix systems, Linux, Windows 2000, and Cray machines. The base LSF product suite consists of LSF Base and LSF Batch. While LSF Base is a stand-alone offering, it must have LSF Batch to provide the sophisticated job scheduling and resource allocation necessary for grids. LSF Base provides the load sharing and distributed processing for the LSF solution suite. It provides services such as host selection, resource information, and transparent remote execution. LSF-based grids can span departments within an organization. AMD has a 1500-node grid that was created using LSF Base and LSF Batch.

LSF Batch is a distributed batch system built on top of LSF Base to provide batch job scheduling services to users. LSF Batch accepts user jobs and holds them in queues until suitable hosts are available. Host selection is a function of up-to-date load information stored in the load information manager (LIM). LSF Batch holds submitted jobs in a job file until conditions are right for them to be executed.

In addition to LSF Base and LSF Batch, LSF companion products include the following:

- *LSF Analyzer* — provides workload analysis across a cluster of computing resources. It generates charge-back accounting reports and removes bottlenecks and helps tune overall system performance.
- *LSF MultiCluster* — supports resource sharing among multiple LSF grids.
- *LSF Make* — dispatches tasks to multiple hosts to reduce job-processing time.
- *LSF Parallel* — manages parallel job execution.
- *LSF JobScheduler* — provides fault-tolerant, scalable, calendar-driven and event-driven scheduling across server grids.

LSF MultiCluster provides for workload sharing across grids by linking queues in distinct grids together. Each companion product has a price tag associated with it.

LSF projects a network of heterogeneous computers as a single system. LSF, like SGE and SGEEE, does not require any alterations to applications. With LSF, jobs

run remotely and behave like jobs run on a local machine. Batch jobs can be run automatically as resources become available. Jobs can be suspended and resumed based on resource availability. In addition, LSF can run sequential and parallel applications and either interactive or batch jobs.

Using LSF, administrators can control access to resources such as the following:

- Who can submit jobs and which hosts they can use;
- How many jobs specific users or user groups can run simultaneously;
- Time windows during which each host can run load-shared jobs;
- Load conditions under which specific hosts can accept jobs or suspend jobs; and
- Resource limits for jobs submitted to specific queues.

LSF supports job checkpointing, permitting job migration to a better host. It supports parallel virtual machine (PVM) and message passing interface (MPI). Several scheduling policies — not to be confused with SGEEE policies — are available for managing batch. These policies include preemptive; preemptable; exclusive; first-come, first-served; and fairshare. Resources can be reserved when a job is submitted, guaranteeing that the job will always have the resources it needs while executing. Or, resources can be gained during job execution, possibly delaying a job's progress during execution because it may have to wait until resources that it needs become available.

Job Flow in LSF

In LSF, a job must be submitted to a queue. A user can name the queue when a job is submitted. When a job is submitted without a queue name, LSF examines the requirements of the job and automatically chooses a queue from a list of default queues. Automatic queue selection is based on user access restrictions (a user may not be permitted to submit jobs to some queues), host restriction (queue must be configured to send the job to all hosts in its specified list), etc.

Queues are not tied to hosts; instead, LSF provides a network-wide view of queues. Jobs can be moved from queue to queue manually. Each time LSF attempts to dispatch a job, it determines which hosts are eligible to run the job. A suitable host is one that has an acceptable load level, has the resource requirements of the job, etc. Each queue has a priority number. Jobs from the highest priority queue are started first. An LSF administrator sets queue priority when the queue is defined. Jobs are dispatched for execution by dispatching those in the highest priority queue first and then in first-come-first-served order within a queue. An administrator can change the order of jobs in a queue.

LSF is designed to be fault tolerant — that is, grid clusters continue to operate even if one or more hosts in the grid are unavailable. Each LSF grid has a master

host that is chosen dynamically. If the current dynamic host becomes unavailable, then another host automatically takes over based on the order in which hosts are listed in the cluster name file.

A job goes through a series of states during its execution phase. Most jobs enter only three states — pend (waiting in a queue for scheduling and dispatching), run (dispatched to a host and running), and done (job completed). A job remains pending until all conditions for its execution are met. Jobs can also be placed in a suspended state by their owners, an LSF administrator, someone with root access, or by LSF itself.

To provide us with your feedback on this research, please go to www.aberdeen.com/feedback

*Aberdeen Group, Inc.
One Boston Place
Boston, Massachusetts
02108
USA*

*Telephone: 617 723 7890
Fax: 617 723 7897
www.aberdeen.com*

*© 2002 Aberdeen Group, Inc.
All rights reserved
May 2002*

Aberdeen Group is a computer and communications research and consulting organization closely monitoring enterprise-user needs, technological changes and market developments.

Based on a comprehensive analytical framework, Aberdeen provides fresh insights into the future of computing and networking and the implications for users and the industry.

Aberdeen Group performs specific projects for a select group of domestic and international clients requiring strategic and tactical advice and hard answers on how to manage computer and communications technology.

About Grid computing



What would it mean if you could:

- Analyze the value of an investment portfolio in minutes rather than hours?
- Unite research teams with others around the world to take advantage of the most up-to-date learnings?
- Significantly accelerate the drug discovery process?
- Scale your business to meet cyclical demand?
- Cut the design time of your products in half while reducing the instances of defects?

Government labs and scientific organizations have been using grid technologies for several years, solving some of the most complex and important problems facing mankind. Now grid computing is becoming a critical component of day-to-day business. Today's challenging business climate requires continuous innovation to differentiate products and services. Businesses must adjust dynamically and efficiently to marketplace shifts and customer demands.

IBM's response to these customer needs is what e-business on demand is all about. There's a profound shift afoot in how computing is used — even in basic assumptions about how it's accessed and paid for. Grid computing can bring tremendous productivity and efficiency to organizations facing the challenges of an on demand world.

IBM has practical information on grid computing

Find out what a grid is.

→ [What is grid computing?](#)

Learn how IBM uses grid.

→ [IBM and grid](#)

Learn about the significant productivity and efficiency gains that grid can offer businesses today.

→ [Grid benefits](#)

Get answers to frequently asked questions for businesses just considering grid computing, as well as those taking the next steps in unleashing grid power.

→ [Frequently asked questions](#)

BEST AVAILABLE COPY

Constructing the ASCI Computational Grid

Judy I. Beiriger, Hugh P. Bivens, Steven L. Humphreys, Wilbur R. Johnson, and Ronald E. Rhea
 Sandia National Laboratories¹, P. O. Box 5800, Albuquerque, NM 87185-1137
 {jibeiri, hpbiven, slhumph, wrjohns, rerhea}@sandia.gov

Abstract

The Accelerated Strategic Computing Initiative (ASCI) computational grid is being constructed to interconnect the high performance computing resources of the nuclear weapons complex. The grid will simplify access to the diverse computing, storage, network, and visualization resources, and will enable the coordinated use of shared resources regardless of location. To match existing hardware platforms, required security services, and current simulation practices, the Globus MetaComputing Toolkit[1] was selected to provide core grid services. The ASCI grid extends Globus functionality by operating as an independent grid, incorporating Kerberos-based security, interfacing to Sandia's CplantTM, and extending job monitoring services. To fully meet ASCI's needs, the architecture layers distributed work management and criteria-driven resource selection services on top of Globus. These services simplify the grid interface by allowing users to simply request "run code X anywhere". This paper describes the initial design and prototype of the ASCI grid.

support higher-level problem solving environments (PSEs). These capabilities include complex work management and resource brokering. This architecture demonstrates the following features:

- The use of Globus services for accessing resources in an independent ASCI environment
- Kerberos-based authentication, grid information authorization and access control
- A Globus interface to the two-tier architecture of Sandia's Computational Plant (CplantTM), a distributed computing resource built of commodity parts
- CORBA-based work management services to support the coordinated use of multiple resources and complex task sequencing
- CORBA-based criteria-driven resource brokering that extends job request/resource matching to support software resources and load-balancing
- Java-based client for generic work requests
- Web-based monitoring
- Integration of the grid infrastructure with CORBA, Java, and Globus-based PSEs and tools.

1. Architecture description

The ASCI grid architecture provides the software infrastructure for an integrated information and simulation environment for the nuclear weapons complex in 2004. The Distributed Resource Management (DRM) project provides grid services to higher level applications. The initial implementation has two development thrusts. First, ASCI requires a core integrated environment for job submission to classified ASCI platforms at the three weapons laboratories: Los Alamos (LANL), Lawrence Livermore (LLNL), and Sandia (SNL). Globus provides these core services, with some extensions for specific ASCI requirements. Second, a software layer above the core services provides a set of common capabilities that

The DRM grid services address three aspects of the shift from platform-centric to network-centric computing: software resources, workflow, and co-scheduling. Software resources serve a large class of users and problems where many resources are suitable, and the primary interest is how soon the results can be obtained. Some simulation codes are run by a number of users at different sites, and must be available for many of the grid computing resources. Multiple versions of these codes are maintained to support different users and to ensure reproducibility of results. Requests to "run code X" are satisfied by software resources and brokering.

Workflow services manage complex task sequencing in the network environment, analogous to the complex scripts often submitted in a platform environment. Workflow can coordinate computational processing steps, computation and visualization, computation and data movement, or other resource usage. Subtasks are scheduled independently.

¹Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Coscheduling of multiple resources will be needed to ensure that the critical resources in the ASCI grid achieve high utilization, to ensure that high priority jobs obtain resources, as needed, and to enable new programming methodologies such as coupled computation and visualization.

2. Related work

The high performance distributed computing community has been researching computational grid concepts and advancing the technology in recent years [2]. The NASA Information Power Grid Project [3], which is advancing grid research into robust operational capabilities, is of particular relevance to the ASCI goal of establishing a grid-based production supercomputing environment. Globus [1] is a collection of infrastructure services that can be used to construct a grid, including services for resource discovery and information, monitoring, security based on the Generic Security System Application Program Interface (GSSAPI), and resource access. Legion [4] is an integrated distributed object computing system that provides security, storage, persistence, naming, and scheduling. Condor [5] finds idle cycles on networked workstations for high throughput applications.

Other researchers focus on software infrastructures that build problem solving environments and tools for parallel applications on top of grid services. Simulation Intranet [6] and WebFlow [7] are CORBA-based environments. Nimrod-G [8] uses Globus services directly. The use of Java for web-based distributed systems is being pursued extensively [9, 10].

3. Conceptual model

The ASCI grid architecture model divides needed software services into several layers and partitions as depicted in Figure 1. The model provides a target of the envisioned 2004 system to support an evolutionary development strategy. It may be abstracted as a three-tier model with domain-specific problem solving environments on top, grid infrastructure services in the middle, and resource interfaces on the bottom. Security and allocation policies must be considered at all layers of the model.

In the top tier, problem solving environments will present users with rich sets of tools and services pertinent to the problem domain. Users can focus on the scientific task, such as a product design or a parameter study, without necessarily dealing with system details such as data locality, resource availability, or platform-specific commands. Developers of PSEs, applications, and tools access the grid services layer for the management and allocation of high performance computing (HPC) resources.

In the middle tier, grid services provide the software infrastructure for accessing geographically distributed resources. These services include discovery, scheduling, reservation, allocation, monitoring, and control of collections of resources. Resource brokering services match resources to user requests based on implicit and explicit constraints. Users must receive consistent, fair, and responsive access to resources regardless of location, while resources must achieve high utilization.

In the bottom tier, an interface layer connects individual resources to the grid. The resources comprising the ASCI grid include heterogeneous computing, communication, storage, visualization, data, and software resources. This layer of the architecture isolates resource-specific interfaces and commands, providing higher level applications with consistent access mechanisms.

Security services and protection mechanisms are needed at all layers of the model. Allocation policies are traditionally implemented for individual resources. To enable coordinated use of resource sets, policy information and services must be available at higher layers.

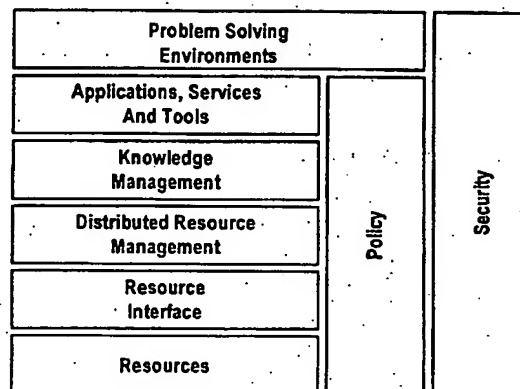


Figure 1. DRM layered architecture.

4. The ASCI grid

A prototype ASCI grid has been implemented in a limited testbed to demonstrate proof-of-concept. It is currently being extended and enhanced as the development environment for operational capabilities. The production grid, illustrated in Figure 2, will connect the laboratories and plants of the DOE weapons complex.

Initial integration prototypes have interfaced grid services to three existing problem-solving environments shown in Figure 3. The Simulation Intranet (SI) is a product design environment that provides web access to simulation, analysis, and visualization tools and manages a product-specific data repository. It is implemented in

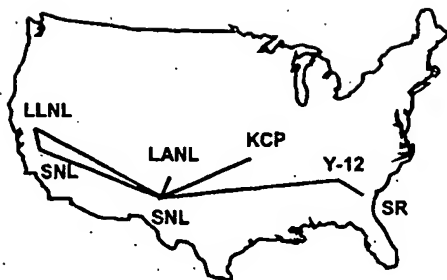


Figure 2. Planned ASCI grid.

Java and CORBA. Focusing on interactive use, the SI environment uses the DRM "run code X anywhere" capability to launch jobs on the most lightly loaded suitable resource. The Integrated Design for Exploration and Analysis (IDEA) environment provides web access to optimization tools for simulation codes. Java-based, IDEA was integrated with the grid monitoring services. Nimrod/G is a parameter study tool that provides its own brokering mechanism and interfaces directly with Globus. It was straightforward to integrate Nimrod-G into the ASCI grid; the same can be expected for other Globus-based tools.

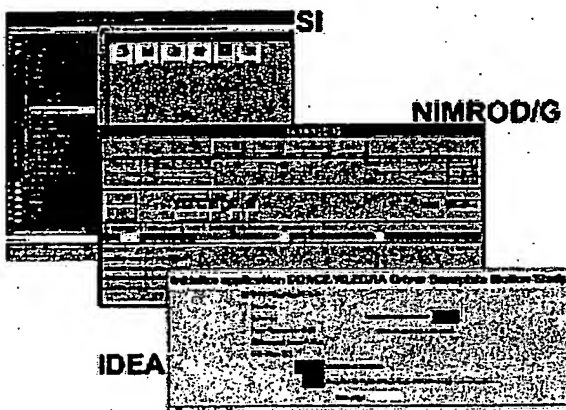


Figure 3. PSEs using prototype grid services.

5. Services for network-centric problem solving

DRM implements a Common Object Request Broker Architecture (CORBA) service layer on top of Globus. CORBA presents a standard way to componentize legacy or enterprise software in a distributed environment. This feature allows distributed deployment of work management and resource brokering capabilities in the grid. In addition, DRM implemented a desktop submission tool in Java for users without specialized

PSEs. The DRM component interactions are shown in Figure 4.

5.1. Work manager

The Work Manager hides the underlying complexity of the grid from scientific users and PSE developers by providing common services for resource usage. The Work Manager is also an autonomous agent that negotiates with the other DRM components to coordinate resource use and executes complex tasks in the ASCI grid on the client's behalf. The Work Manager accepts job requests, or work specification, from DRM clients. This work specification is represented as an eXtensible Markup Language (XML) document and provides users and PSEs with a grid-level "scripting" language. The work specification can consist of computations, file migrations, resource attributes and constraints, and task sequencing. A DRM client application creates a Work Manager on the server side and sends the work specification using the *submit* method of the Work Manager's CORBA API. The Work Manager generates a query to the Resource Broker, which returns the best match. The Work Manager uses the query results to construct a specific resource request in the Globus Resource Specification Language (RSL). Using the Globus Resource Allocation Manager (GRAM) Client API, the Work Manager submits the request to the target resource. The Work Manager receives status updates and redirected standard out and standard error streams from the GRAM and forwards these to the DRM client.

The Work Manager design supports the coordinated use of multiple resources. The initial prototype implements services for single resource requests and simple job management. It is currently being extended to support coallocation, serial, parallel, and conditional task sequencing. Lead-lag and deadline dependencies will be added as scheduling services are incorporated into the ASCI grid.

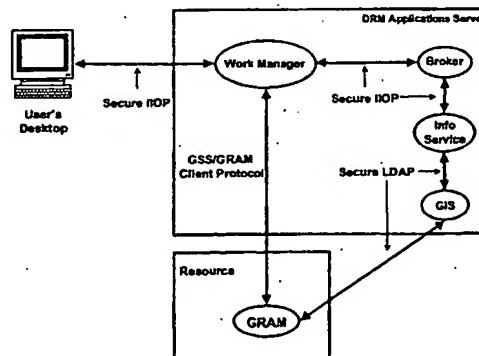


Figure 2. DRM's distributed object architecture.

5.2. Resource broker

The purpose of the Resource Broker is to select the "best" resource, allowing the user to specify only the minimum parameters of interest. The Broker locates the set of resources that the user can access and satisfy the attributes of the request. From the set of possible resources, the Broker chooses one based on a small number of possible selection algorithms. With a default selection algorithm, it is possible for the user to merely specify "run code X". This contrasts with the common practice of selecting a machine or logging in directly, where the user must have prior knowledge of the resource. It also contrasts with the concept that a service is a software component installed on a resource [6], where again the user must have specific knowledge.

The Resource Broker first queries the grid information service (GIS) for a set of resources that satisfy the request. A CORBA interface to the LDAP-based GIS has been developed to support the Broker queries. The Broker then filters the set of resources according to the criteria in the request. Requests can specify a particular resource (called a "white pages" request), or can specify criteria that the resource attributes must satisfy (called a "yellow pages" request). The conditional operators ($=$, $<$, $<=$, $>$, $>=$) can be applied to values for numeric attributes, and the logical operators ($\&$, $+$, $!$) can be applied to combine attribute criteria. Any object in the GIS – a person, an organization, software, hardware – can be brokered.

Several examples illustrate how request criteria are applied. The request "cn=blue-mountain.asci.lanl.gov-lsf" specifies a particular machine, the ASCII SGI platform at Los Alamos. The request "(&(&(osname=irix)(freenodes>=32))(!(&(hn=atlantis.sandia.gov)))" matches any SGI machine (including Blue Mountain) with at least 32 available nodes, but specifically rejects Atlantis. The request "sw=Alegra-2D" identifies all machines that can run that code. The Broker treats the user identity as an implicit constraint, filtering out resources where the user is not authorized.

Once a set of suitable resources is identified, the "best" one is selected. The DRM Resource Broker will provide a small number of selection algorithms that will serve most requests. An initial prototype algorithm was implemented to support interactive use by choosing the least loaded resource. This can also provide load balancing. For each suitable resource, the Broker calculates a load factor by weighting the CPU loads for the previous one, five, and fifteen-minute intervals and averaging against the CPU clock speed. The maximum load factor corresponds to the least loaded resource. The Broker returns to the requestor all of the attribute/value pairs in the GIS for this resource.

The second selection algorithm is being prototyped for a queuing environment by choosing the resource with the least average wait time for a given job type. The job type

maps all possible job requests to a set of characteristic job types for which historical average wait time data will be maintained. Parameters that affect how long a job waits in a queue include the resources requested, such as number of processors and time, and the relative priority of the request with respect to competing requests. Requests can be prioritized according to the fair share concept of a service ratio (SR). The SR is implemented differently at each site, but in all cases is a normalized measure of how much of the resource a user has consumed relative to how much the user is entitled to. The effects of competing requests have been observed in cyclic patterns in the workload characteristics for a resource, such as the pattern of requests and behavior over a typical week of workday and night and weekend shifts.

For this selection algorithm, the job type will consider the following parameters: number of processors, time requested, SR, time of day, and day of cycle. For each parameter, the possible range of values is divided into a set of subranges, or bins. For example, time requested could be binned into 15-minute intervals, SR into tenths, and time of day into 1-hour intervals. An individual job request maps to a particular 5-tuple of parameter bins. A historical record of the average wait time for each 5-tuple will be maintained by the resource. The Broker will get the SR for the user, and then the average wait time for the job type. The average wait time will be adjusted for data movement and non-routine planned outages. The resource with the least predicted wait time will be selected.

The Resource Broker is being extended to support requests for collections of resources. An algorithm for closest match selection is also desired. When a resource request can not be satisfied, the closest match selection will attempt to identify a resource that is similar to the request.

5.3. Desktop submission tool

A desktop submission tool was developed for users without domain-specific PSEs. This is a Java-based GUI for obtaining DRM services. The desktop submission tool creates a work manager and sends it a work specification. The initial implementation allows a user to input a description for reference and to specify or select an application. Optionally, the user may input command line arguments, request input file staging via text entry or browsing, or specify resource constraints. Then the user may submit the job request, monitor job status, and display standard out and standard error streams that are sent back from the work manager. The tool is being extended to support requests for new types of resources like network bandwidth, collections of resources, and complex task sequencing. The initial prototype is shown in Figure 5 and is an example of a physics code running after submission to DRM.

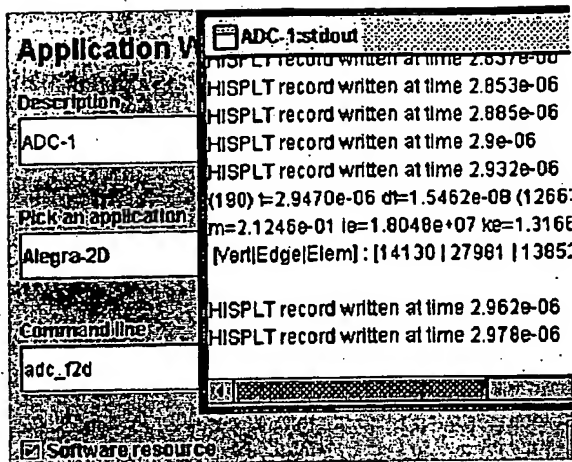


Figure 3. The desktop submission tool.

6. Extensions to Globus

The ASCI environment requires extensions to Globus services. First, the Globus Resource Allocation Manager (GRAM) must be extended for computing resources with unique features. This ability was demonstrated by interfacing Globus to CplantTM. Second, monitoring extensions display node status and job attribute information. Third, ASCI requires Kerberos-based security services.

6.1. CplantTM resource interface

One of the primary reasons that the Globus toolkit was chosen as the foundation of the DRM architecture was its ability to connect to existing resource management systems. The ASCI grid contains both resources for which GRAMs exist, and unique local managers that the Globus group has not encountered before. Fortunately the design has proven robust and accommodates these differences through a flexible data model.

The grid data model has two basic aspects that must be carefully designed and managed. First is the systems view of the grid: representations of resource attributes and state for efficient operation of grid applications. Second is the end user view of the grid: a collection of computational resources. These two notions are not always in agreement. One example is the CplantTM architecture, which has multiple service nodes (front ends), a unique allocator called yod, internal and external network interfaces, and a customized version of Portable Batch System (PBS) as a local resource manager.

Because of the size of CplantTM, the number of users accessing it at any given time is large. Load balancing user connections today is achieved by cycling service node network names from a common name using the

domain name system (DNS). This works well in a login environment since the users only have to know the common name, but quickly breaks down in a grid environment, as DNS will be unreliable due to intermediate lookups. The goal is to model the resource as a single system (the user view), while also describing the individual service nodes to Grid applications. In Globus this requires changes to the GRAM reporter and site-build daemon. Coordination of reporting features will eliminate duplicate information being written to the GIS and fail-over semantics will permit continued operation should a reporting service node die.

The yod allocator assigns nodes in an order that is efficient to the topology of its internal Myrinet interconnect. However, as a CplantTM system grows, the compute nodes will not necessarily remain homogeneous. Disparate memory sizes, network speeds, and local disk space will determine whether a given set of nodes is desirable to the user. To request a suitable set of nodes in a heterogeneous system, changes to the node allocator, the GRAM site-build, GRAM reporter daemons, and job-manager are required.

Node allocator changes will expose the differences between compute nodes in such a way that it is possible to identify the varying attributes associated with each node and to specifically request a set of nodes. This will require that either a node identifier is bound to each compute node in the system, or a class of nodes is defined. At the time of job submittal, a set of identifiers or a node class is passed to the allocator. This approach also creates problems with the CplantTM batch queuing system because the scheduler will have to discriminate which jobs want what nodes and when are they available.

The GRAM site-build daemon will have to be modified to report node configurations. The ability to discover node identifiers/classes and their associated attributes will allow the site-build daemon to report the current CplantTM system configuration into the GIS, making them available to the Resource Broker. The GRAM reporter will need to report node consumption for brokering and monitoring purposes. Currently, Globus only reports the numbers of total and free nodes in such a system.

Because of changes to the data model, the job manager must know that it is running on a service-node and not on the abstract DNS definition (user view) in order to report the correct global job id back to the GRAM client. This is accomplished by ensuring that the domain name defined at the service node does not reference any of the other service nodes in the system.

The PBS installation on CplantTM is also customized. Therefore, it is important to incorporate native management capabilities, most notably the pingd utility. A hook into the GRAM reporter provides information on CplantTM node usage back to the GIS. In addition, because

Cplant™ PBS uses yod features, the PBS_NODES reference in submittal scripts is currently unnecessary.

These architecture features are not unique to Cplant™. They exist in other ASCI resources and any complex installation with multiple front ends and heterogeneous resources.

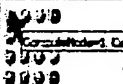
6.2. Monitoring services

Monitoring services are presently supported through two mechanisms; the Heart Beat Monitor (HBM) and the myriad number of grid services that report status and information to the GIS. The HBM is a collection of small-footprint Unix daemons provided as part of the Globus grid computing toolkit. Local monitors on each of the grid-enabled resources and the DRM server periodically perform the Unix *ps* command, parse the output, and determine the status of the software services that have registered as HBM clients. Each local monitor then reports to the HBM data collector (located on the DRM server) the status of its clients. The information reported by the local monitors is limited to one of four states of health. The HBM client is reported as being either active, overdue (hasn't shown up in the most recent *ps* output), down, and unknown.

In contrast to the limited amount of information provided by the HBM, the GIS provides a wealth of information about the compute resource, queue status, job performance, and network status. Most of this information is obtained by the GRAM reporter that runs on each resource for each local resource manager (fork, PBS, NQS, DPCS, LSF, etc.). Other resource specific information is reported by the site-builder, a Globus provided Unix daemon that reports relatively static resource data. End-to-end network bandwidth and latency are reported by gloperf, another piece of Globus software that provides a grid-enabled interface to netperf, an open source network performance monitor.

All of the above monitoring data is available to the user through a collection of web pages that access, parse, and present the data using CGI scripts which run on a

Node Status for service-11.sandia.gov-fork

	ScalableUnit 11
Status	

Legend: [] Free, [] Used, [] Unknown

Display Node Attributes Table

Figure 4. Web-based node status monitoring

secure web server. The use of a web-based interface allows for the easy modification of how the information is presented. Figure 6 shows an example of CGI script customization that was used to display individual node status of ASCI's Cplant. Colored balls represent compute nodes. Yellow nodes are allocated, and display job information when the mouse cursor is placed over the node representation.

6.3. Security services

The ASCI grid must support classified computing in compliance with various Federal and DOE regulations. The grid services must communicate securely over multiple networks and between numerous compute resources. In the ASCI grid, Kerberos Version 5 is the primary mechanism for authentication. Kerberos and several other mechanisms are also used for protecting (encrypting) the data as it moves from grid service to grid service. The Sandia developed Generalized Security Framework (GSF) will be used to provide an abstraction layer over the Kerberos and other security libraries [11]. Both Java and C++ applications can use the GSF.

Users can connect via Secure Shell (ssh) to the DRM server or run grid enabled PSEs, frameworks, and applications, from their desktops. Grid-enabled software, such as the DRM desktop submission tool, will establish authenticated and protected connections between user work stations and the DRM server using GSF secured Internet Inter-Orb Protocol (IIOP) connections. Users can also access a secure web server, to monitor the status of grid resources and submitted jobs.

The DRM Work Managers, Resource Broker, Information Service, and GRAM Clients will initially all be located on the same server. Connections between these services will be authenticated using Kerberos. The GIS will also be located on this server. Access to the GIS data must be authenticated and authorized. Access to the GIS LDAP server will be authenticated using a Netscape Directory Service (NDS) plug-in that provides a Simple Authentication and Security Layer (SASL) mechanism for authentication using GSSAPI over Kerberos. Access will be authorized using standard LDAP Access Control Instructions (ACIs).

Connections between the GRAM Clients and GRAM Gatekeepers will almost always occur across one or more network connections. Authentication will be accomplished using GSSAPI over Kerberos. Data protection will be achieved by using the Secure Sockets Layer (SSL) protocol. The remaining GRAM processes will use authenticated inter-process connections. Data transfers using the Kerberos authenticated and SSL protected Globus Access to Secondary Storage (GASS) will also be supported.

7. Current status

An initial ASCI grid exists in a Tri-Lab testbed, and prototype grid services were demonstrated at SC99. The initial grid services include a core Globus-based environment for job submission and monitoring, and a layer of CORBA-based services, such as work management and resource brokering that support network-based problem solving methodologies. Current activities are focused on the security and robustness needed for production use. Fully integrating Kerberos into Globus is needed to obtain DOE security accreditation for use in classified networking environments. The robustness of the grid information service is being enhanced through replication and referral. DRM is scheduled to be deployed on ASCI White, the new 10 Tops IBM supercomputer at LLNL, in November 2000.

8. Future work

DRM capabilities that provide sophisticated services for network-based problem solving will be added to the ASCI grid. Advance reservation, coscheduling, and workflow will allow scheduling and coordinated access to storage systems, network bandwidth, and visualization resources. A scheduler component will be added to the DRM architecture to realize grid-level scheduling. The Work Manager, Resource Broker, and GIS must also be extended to support reservations and requests for collections of resources. DRM will continue to collaborate with PSEs and participate in the Grid Forum.

9. Conclusion

DRM is developing and deploying a computational grid to support the ASCI mission. The Globus resource management infrastructure provides a highly sophisticated and extensible mechanism for adding and exploiting new and existing ASCI resources. The grid's CORBA layer on top of Globus aids scientists and problem solving environments by simplifying job management and resource discovery. All grid services communicate securely and access to the grid information service is access controlled. With these features, the ASCI grid will offer improved utilization of geographically distributed HPC resources in the Tri-Lab complex.

10. References

- [1] Foster, I., and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 1997
- [2] *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, ed., Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1999.
- [3] Johnston, W., Gannon, D., and Nitzberg, B., Grids as production Computing Environments: The Engineering Aspects of NASA's Information Power Grid, in *Proceedings of Eighth International Symposium on High Performance Distributed Computing*, August, 1999.
- [4] Grimshaw, A., W. Wulf, and The Legion Team, The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 1997
- [5] Litzkow, M., M. Livny, and M. Mutka, Condor - A Hunter of Idle Workstations, in *Proceedings of the 8th International Conference Of Distributed Computing Systems*, June 1988
- [6] Holmes, V., J. Linebarger, D. Miller, and R. Vandewart, 1999, The Simulation Intranet Architecture, in 1999 *International Conference on Web-Based Modeling & Simulation*, San Francisco, CA, January 17-20, 1999.
- [7] Akarsu, E., G. C. Fox, W. Furmanski, and T. Haupt, WebFlow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing, *Proceedings of SC98*, 1998.
- [8] Abramson D., R. Sosc, J. Giddy, and B. Hall, Nimrod: A Tool for Performing Parameterised Simulations using Distributed Workstations, *The 4th IEEE Symposium on High Performance Distributed Computing*, August 1995.
- [9] Baratloo, A., M. Karaul, Z. Kedem, and P. Wyckoff, Charlotte: Metacomputing on the Web, *Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.
- [10] Christiansen, B., P. Cappello, M. Ionescu, M. Neary, K. Schauser, and D. Wu, Javelin: Internet-based Parallel Computing Using Java, Department of Computer Science, University of California, Santa Barbara, 1998.
- [11] Detry, R., Kleban, S., Moore, P., and Berg R., The Generalized Security Framework, Presented at CSCoRE 2000, <http://www.ccs.bnl.gov>, Brookhaven National Laboratory, NY.

Application-Level Scheduling on Distributed Heterogeneous Networks * (Technical Paper)

Fran Berman and Rich Wolski[†]
Silvia Figueira, Jennifer Schopf, Gary Shao
Department of Computer Science and Engineering 0114
University of California, San Diego
La Jolla, Calif. 92093 [‡]

Abstract

Heterogeneous networks are increasingly being used as platforms for resource-intensive distributed parallel applications. A critical contributor to the performance of such applications is the scheduling of constituent application tasks on the network. Since often the distributed resources cannot be brought under the control of a single global scheduler, the application must be scheduled by the user. To obtain the best performance, the user must take into account both application-specific and dynamic system information in developing a schedule which meets his or her performance criteria.

In this paper, we define a set of principles underlying application-level scheduling and describe our work-in-progress building AppLeS (application-level scheduling) agents. We illustrate the application-level scheduling approach with a detailed description and results for a distributed 2D Jacobi application on two production heterogeneous platforms.

1 Introduction

Fast networks have made it possible to coordinate distributed CPU, memory, and storage resources to provide the potential for application performance superior to that achievable from any single system [1]. Parallel applications targeted to such systems are typically resource-intensive, i.e. they require more resources than are available at a single site [16]. Critical resources may include large aggregated and distributed memory, fixed data sources,

The authors were supported in part by NSF grants ASC-9301788, ASC-9308900, and a scholarship from CAPES and UFRJ (Brazil).

Presenting author.

Email addresses of the authors are {berman, rich, silvia, jenny, gshao}@cs.ucsd.edu. Fran Berman's phone is 619-5346195 and fax is 619-5347029.

local temporary storage, and computational cycles. Performance is defined by the user, and may mean different things for different applications, however achieving it requires the efficient use of all relevant resources.

Despite the performance potential that distributed systems offer for resource-intensive parallel applications, actually achieving the user's performance goals can be difficult. One of the most fundamental problems that must be solved to realize good performance is the determination of an efficient schedule. Effective scheduling by the application developer or end-user involves the integration of application-specific and system-specific information, and is dependent on the dynamic interactions between an application and the relevant system(s).

Currently, the performance-seeking end-user must develop schedules for distributed heterogeneous applications off-line, using intuition to predict how the application will perform at the time it will execute. The users or application developers must select a configuration of resources based on load and availability, evaluate the potential performance of their application on such configurations (based on their own performance criteria), and interact with the relevant resource management systems in order to implement the application. At the same time, other users (running their own applications) draw from the same set of resources, each seeking to achieve his or her own performance goals. When multiple users contend for resources, only a fraction of the resource performance can be delivered to each.

In this paper, we describe an application-specific approach to scheduling individual parallel applications on production heterogeneous systems. We are developing software to facilitate and improve upon the scheduling activities of the user. Our goal is to develop scheduling agents that perform this task for the user at machine speeds and with more comprehensive information. We term these agents **AppLeS – Application-Level Schedulers**. Each application will have its own AppLeS to determine a performance-efficient schedule, and to implement that schedule with respect to the appropriate resource management systems.

Note that AppLeS is not a resource management system; rather, it interacts with systems such as Globus [3, 11], Legion [12, 17], or PVM [9, 20] to perform that function. As such, AppLeS is an **application-management** system which manages the scheduling of the application for the benefit of the end-user.

In the next subsection, we describe our approach for AppLeS.

1.1 Scheduling from the Perspective of the Application

Application-level scheduling is based on four underlying principles:

- **Application- and system-specific information is needed for good schedules.** Users determine good schedules for their applications based on their perception of system capabilities, and their knowledge of the structure and requirements of their application. The frequency of communication and computation, the amount of memory required, the number, type, and size of application data structures are matched with the granularity of the computational platforms, network speed and bandwidth, and other system attributes to develop a performance-efficient schedule.

- **Dynamic information is necessary to determine system state.**

Users base candidate schedules on knowledge of which machines are available and which are heavily or lightly loaded. This load varies over time and with usage of system resources. If a choice of networks or computational platforms is available, the user will combine his/her knowledge of how the application will use the system with the current or predicted load on its resources.

- **Good schedules involve some prediction of application and system performance.**

Prediction provides the basis for most scheduling. The user predicts how their application will execute on the system and uses this prediction to choose a performance-efficient schedule. Such predictions are difficult to make accurately since the system varies over time due to contention, and application performance may be dependent on both data and system load. However, simplifying the model of the system or application excessively to make the prediction task easier is not always fruitful. Optima for a simplified model may not correlate with optimal behavior in practice. In particular, application and system models must be sufficiently complex to expose real phenomena.

- **All resources can be evaluated strictly in terms of the performance they deliver to the application.**

Notice that, from the perspective of the application (or user), each resource is judged ultimately on how much it benefits the application's execution. Users define different criteria for performance (speed, cost, etc.), but the decision about which resources to use, and when to use them, is based on how they will perform (in terms of the specific criteria) when executing the user's application.

The AppLeS approach is to use parameterizable application- and system-specific models to predict application performance using a given set of resources. Using these models in conjunction with forecasts of expected resource load, an AppLeS agent can select a resource set and an application schedule by evaluating various candidate mappings. The mapping that generates the best expected performance is chosen and implemented on the target resource management system(s).

Note that a fundamental difference between the AppLeS approach and system-oriented schedulers is that for AppLeS, **everything about the system is experienced from the point of view of the application**. If the candidate resources for the application are lightly loaded, then the system appears lightly loaded to the application regardless of the load on other resources. If the candidate resources are heavily loaded, then the system appears heavily loaded.

In the next section, we utilize the application-level scheduling approach to develop an efficient schedule for a distributed Jacobi data-parallel code. The example serves as a "proof of concept" for the principles underlying the AppLeS approach, and serves to illuminate the components required for general application-oriented scheduling agents. After discussing the Jacobi example in detail, we will describe our current efforts to build general AppLeS agents for scheduling in Section 4.

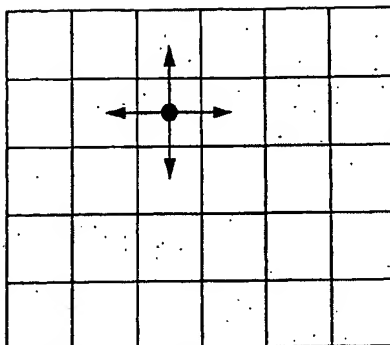


Figure 1: Five-point Jacobi Computation

2 Application-Level Scheduling of Jacobi2D

In this section, we illustrate and motivate our approach using a simple application. We discuss the development of an application-level schedule for a distributed 2D Jacobi application in detail and present performance data.

Consider the problem of executing a distributed data-parallel two dimensional Jacobi iterative solver (Jacobi2D) using a heterogeneous network of machines. The Jacobi method is commonly used to solve the finite-difference approximation to Poisson's equation [15] which arises in many heat flow, electrostatic, and gravitational problems. Variable coefficients are represented as elements of a two-dimensional grid. At each iteration, the new value of each grid element is defined to be the average of its four nearest neighbors during the previous iteration (see Figure 1).

Typically, the Jacobi computation is parallelized by partitioning the grid into rectangular regions, and then assigning each region to a different processor. This decomposition strategy is favorable because a processor need only obtain the border elements for its region during each iteration. The amount of computational work scales as the area of each region, whereas the amount of delay due to communication scales as the perimeter. A small number of big regions will yield good processor efficiencies, but may sacrifice parallelism. Conversely, a large number of small regions may incur large communication overhead. In our example, the user wishes to identify a partitioning that yields the lowest possible execution time. Solving the partitioning problem optimally is NP-complete, so it is necessary for the user to employ heuristics to arrive at a "good" solution.

2.1 Deriving Partitions that Optimize Resource Performance

The version of Jacobi2D we use in this example is written in a data-parallel SPMD style using KeLP [6, 5]. The KeLP system provides high-level abstractions, in the form of C++ objects, that support runtime data decomposition. In addition, the details associated with message passing in distributed-memory computing environments are buried in the abstrac-

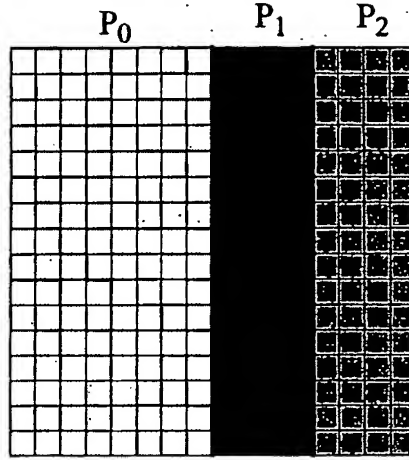


Figure 2: Strip data partitioning for three processors where processor P_0 is twice as fast as processor P_1 or P_2 .

tions making the code portable and easy to maintain.

An ideal partitioning will assign regions of the Jacobi grid to processors such that the area of each region matches the performance capability of the processor to which it is assigned. Faster processors should compute over larger regions than slower ones. In particular, computational time is optimized when the ratio of each rectangular area of the grid to the total grid area most closely matches the ratio of the power of the processor to which the rectangular area is allocated to the total processing power available.

However, it is not simply a processor's computational time that defines its performance capability for Jacobi2D. The performance capability of each processor depends on how fast each processor can locally compute an element of the Jacobi matrix, and how quickly each processor can communicate its border elements with its neighboring processors. These two factors most dramatically affect execution time of this application.

To derive partitions that balance resource performance, we formulate the partitioning problem as an analytical model. Let

T_i = time for processor i to compute region i

A_i = the area of region i

P_i = the time required for processor i to compute a single point locally

C_i = the time for processor i to send and receive its borders

for i in I regions and processors. The time each processor spends computing and communicating during a single iteration of Jacobi2D can then be represented as

$$T_i = A_i * P_i + C_i$$

This equation predicts the execution time (including the time spent communicating) for each processor. If all partitions are scheduled simultaneously, then the execution time for a single iteration will be equal to the maximum value of T_i . We can balance the time each processor spends computing and communicating by setting all T_i equal and solving the resulting system of equations for A_i . For a grid with N rows and M columns, let

$$\begin{aligned} T_1 = T_2 = T_3 = \dots = T_I \\ \sum_{i=1}^I A_i = N * M \end{aligned} \quad (1)$$

We restrict the legal partitions to those which only consider a single dimension (i.e. strip partitions, shown in Figure 2), so that C_i does not depend on A_i . For this type of partitioning, the system of equations (1) is linear and can be solved quickly by conventional methods.

For a strip partitioning, we let

$$C_i = \begin{cases} Recv(i-1, i) + Recv(i+1, i) + Send(i, i-1) + Send(i, i+1) & \text{for } i \geq 2 \text{ and } i \leq (I-1) \\ Recv(i+1, i) + Send(i, i+1) & \text{for } i = 1 \\ Recv(i-1, i) + Send(i, i-1) & \text{for } i = I \end{cases}$$

where $Recv(i, j)$ = time to receive N elements from processor i on processor j

$Send(i, j)$ = time to send N elements from processor i to processor j

N = number of elements in the dimension not being partitioned

We can solve the linear system of equations (1) in $O(I^3)$ by simple Gaussian elimination for each A_i . Note, however, that there is no guarantee that each A_i corresponds to an integral number of columns (or rows). To complete the strip decomposition, we must then round the partitions accordingly.

Observe that an alternative, but computationally more complex, solution is to formulate the problem as a constraint-based minimization problem. Linear programming techniques can then be used to derive the partitions. This approach is viable, however in the interest of rapid prototyping, we chose to adopt the simpler linear systems formulation.

2.2 Predicting System State with the Network Weather Service

To solve the linear system of equations (1), we require as parameters the time required to send and receive N elements from each processor to its neighbors ($Send(i, j)$ and $Recv(i, j)$), and the time required to compute a single element on each processor (P_i).

We can model the send and receive times as

$$Send(i, j) = N * sizeof(element) / Bandwidth(i, j)$$

$$Recv(i, j) = N * sizeof(element) / Bandwidth(j, i)$$

where $Bandwidth(i, j)$ = data rate supported by the link between i and j

Note that N and $\text{sizeof}(\text{element})$ are both time-invariant parameters of the problem being solved. Similarly, we can model the per point compute time on each processor i as

$$P_i = P_{\text{Unloaded}_i} / \text{CPU}_i \text{ where}$$

P_{Unloaded_i} = the time to compute a single point on an unloaded processor i , and

CPU_i = the percentage of time processor i spends executing partition i

These quantities will vary over time due to resource contention. $\text{Bandwidth}(i,j)$ will be defined (in part) by the volume and frequency of traffic crossing the link from i to j . CPU_i will depend on the number of additional processes executing on processor i , and the way in which each CPU is managed. Typically, if the system is time shared, the percentage of time a CPU is devoted to any one job is some "fair share" of the total CPU time; however, that share will change as jobs enter and leave the system.

Moreover, the estimates of $\text{Send}(i,j)$, $\text{Recv}(i,j)$, and P_i must be accurate at the time the application will be scheduled which is not necessarily the time at which the partition is derived. The scheduler, therefore, requires a forecast of the values of $\text{Send}(i,j)$, $\text{Recv}(i,j)$, and P_i for the time frame in which the application will execute.

We have developed a separate facility called the Network Weather Service which dynamically supplies values and forecasts for CPU_i for all i , and $\text{Bandwidth}(i,j)$ for all i and j in a networked system. The Network Weather Service is outlined in Section 4. For Jacobi2D, the Network Weather Service used dynamic probes and load history to help forecast CPU_i and $\text{Bandwidth}(i,j)$ at the time the application was to be scheduled.

2.3 Resource Selection and Scheduling

Resource selection focuses on the identification of a subset of resources that most efficiently support the application. Most users naturally focus on resources they perceive as being "close". For the Jacobi application, we can formally define the logical "distance" between resources and prioritize a resource set based on this metric. Note that distance between resources is meaningful to the application only in terms of how the resources will be used. Recall that for a given grid region of size N^2 , the computation in each partition scales as $O(N^2)$ and the communication scales as $O(N)$. We can use this relationship to define the distance between processors for Jacobi2D. Let

P_i = the forecast time required for processor i to compute a single point locally

$\text{CE}(i,j)$ = the forecast time for processor i to send and receive a single element to and from processor j

Then

$$D(i,j) = N^2 * (|P_i - P_j|) + N * (\text{CE}(i,j) + \text{CE}(j,i))$$

defines a distance measure between processors i and j for a arbitrary problem size N . Two processors are near to each other in Jacobi2D if their compute capabilities are relatively equal, and if their interprocess communication is fast.

To select resources from the global resource pool, we start by identifying a candidate machine to serve as the locus. For example, the user's machine or the fastest machine in a cluster may serve as the locus. The rest of the machines are then sorted according to their distance D from the locus. Note that different orderings may be determined for distinct loci. The first K elements of the sorted list for a particular locus L are defined to be the "closest" resource set to L containing K machines.

For Jacobi2D, the workstation with the fastest CPU was used as one such locus. We then used the algorithm in Figure 3 to determine a candidate resource set.

```

let head = locus
let tail = locus
for i in 1 to I-1
    find the machine m such that  $D(\text{tail}, m)$  is a
        minimum and m is not already on the list
    add m to the tail of the list
    let tail = m
end

```

Figure 3: Prioritizing the resources based on "distance".

```

let locus = machine having the maximum criterion value
let list = a sort of the remaining machines according to
    their logical distance
for k in 0 to I-1
    let S = {locus + the first k elements of list}
    parameterize  $C_i$  and  $P_i$  for  $1 \leq i \leq |S|$  with
        Weather Service forecasts
    solve linear system of equations using this parameterization
    if(not all  $A_i > 0$ )
        reject partitioning as infeasible
    else if(there exists an  $A_i$  that does not fit in free memory
        of processor i)
        reject partitioning as infeasible
    else record expected execution time for subset S
end
implement, the partitioning corresponding to the minimum
    execution time using the S for which it was computed

```

Figure 4: Resource selection and scheduling algorithm for Jacobi2D.

The algorithm iteratively finds the machine that is closest to the current tail, and adds that machine at the tail end of the list. After all I machines have been added, the algorithm

terminates with each machine logically closest to those adjacent to it in the list. This form of sorting is useful for a strip decomposition of Jacobi2D as processors only communicate with at most two neighbors.

Having derived the resource list, the Jacobi2D scheduler then proceeds to compare different potential partitionings using subsets of the total list. It starts by estimating the execution time on the locus machine. Next, it considers a two processor partition using the first two processors on the list. It parameterizes the linear system of equations for $I=2$ processors, and consults the Network Weather Service for the performance forecasts that pertain to those two machines. After solving the linear system, it records the estimated execution time of the resulting partition. A three processor partitioning using the first three processors from the sorted resource list is considered next. The estimated execution time for the three processor system is recorded, and the algorithm continues until all processors from the list are considered or some predefined maximum logical distance from the locus is reached. Finally, a processor set and a partitioning and schedule yielding the minimum estimated execution time are chosen as the "best" schedule for that locus. Note that when $I=1$, the Resource Selector considers a single-site implementation. In our example, the single-site implementation is simply a sequential version of the KeLP implementation. If an optimized implementation for a particular system were available, the Resource Selector could consider that as well.

Each time a partition is generated in the process, it is checked for feasibility. Two filters are employed to remove infeasible partitions from those ultimately considered for scheduling. The first filter removes partitions that have negative values of A_i . These correspond to mappings where the communication time is so great, the processor must compute a negative number of elements (implying a negative execution time) in order to finish with the other processors. The second filter checks to make sure that the size of each partition fits within the free memory (forecast by the Network Weather Service) available on the machine to which it is assigned.

The resource selection and scheduling method used by our example Jacobi2D scheduler can be summarized by the pseudocode in Figure 4.

2.4 Scheduling Jacobi2D and the AppLeS PrincipLeS

The scheduling approach we have described for Jacobi2D uses the principles outlined in Section 1.1 and in fact is an example of an AppLeS. Application-specific and system-specific information are used throughout the scheduler, both to generate schedules and to select resources. Dynamic system information is provided via the Network Weather Service to parameterize performance models. Predictive models are used to evaluate and rank candidate schedules. Finally and perhaps most important, all resources are considered strictly in terms of how they affect application performance.

Using this application-level approach to scheduling, the natural question becomes "How performance-efficient is the schedule that it generates?" We describe experiments which address this question in the next section.

3 Performance Results for the Jacobi2D Application-Level Schedule

To determine the effectiveness of the application-level scheduling approach, it is important to answer the following questions:

- How does the execution time of Jacobi2D using an AppLeS schedule compare to a schedule determined using a widely-accepted conventional method?
- What is the effect of using dynamically forecast resource performance data in the application-level scheduling approach?
- What is the effect of automatic resource selection in the application-level scheduling approach?

To address these questions, we compared four partitioning methods for the same KeLP implementation of Jacobi2D. The first method [**Compile-time blocked**] uses a conventional HPF-style [14] block partitioning in which each processor is assigned (at compile-time) a relatively equal-sized square region of the grid to compute. The other three partitioning methods utilize versions of the application-level scheduling approach described in the previous section. Partitioning method 2 [**Compile-time AppLeS**] uses good static estimates for resource performance and uses resource selection to select a resource set from the total resources. Partitioning method 3 [**Runtime AppLeS/No Select**] uses dynamic estimates from the Network Weather Service for resource performance but assumes that the user wants to use all available resources. Partitioning method 4 [**Runtime AppLeS**] uses dynamic estimates and resource selection – it constitutes the full application-level scheduling approach discussed in the last section. Note that partitioning methods 3 and 4 utilize Network Weather Service data and so must be performed at run-time, whereas partitioning methods 1 and 2 use static data and may be performed at compile-time.

All four versions first partition and distribute the grid, and then execute the Jacobi solver. That is, the data and computations are scheduled on the processors once before execution begins, and remain there for the duration of the execution. We are currently formulating a version of the Jacobi application-level scheduler which effectively redistributes the grid in response to changing load on system resources. This flexibility is supported in the AppLeS software described in the next section.

3.1 Execution Performance

To investigate the relative execution performance of the four partitioning methods, we used eight non-dedicated workstations located at the San Diego Supercomputer Center (SDSC) and the U.C. San Diego Parallel Computation Laboratory (UCSD-PCL). The workstation set consisted of a Sun Sparc-2, a Sun Sparc-10, and two IBM RS6000 workstations located at UCSD, and four DEC Alpha workstations located at SDSC. Numeric format conversions were handled by KeLP which uses MPI as its underlying communication substrate. The network connecting these systems was also heterogeneous and non-dedicated. Within the

PCL, the Suns were attached to an ethernet segment shared by several other systems. The RS6000s were connected to a different segment (also shared by other ambient machines) and a gateway which linked the two segments. At SDSC, the Alpha workstations were connected to non-dedicated FDDI ring. The configuration is shown in Figure 5.

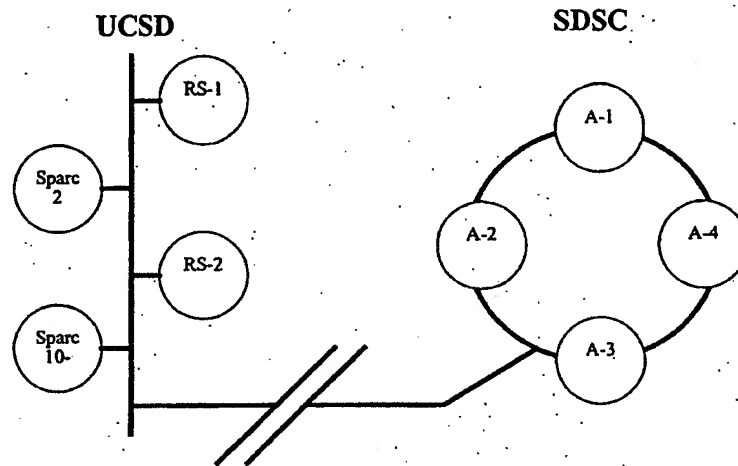


Figure 5: Workstations and Networks used at UCSD and SDSC.

All systems and networks were shared and used in "production mode" while we ran our experiments. Since conditions might change between one execution and the next (due to contention) we made several runs for each problem size, and reported the average execution time of a single iteration. During each experiment, we ran one instance of each of the four partitioning methods back-to-back hoping that all four executions would enjoy similar conditions, on average. Figure 6 shows the average iteration execution times (in seconds) for a range of problem sizes. In each case, a square grid having the problem size dimension shown in the figure was used.

In the experiments, application-level scheduling is able to outperform the block partitioning because it uses its performance model to predict how well each resource will perform when executing a piece of Jacobi2D. It uses that prediction to determine how much of the grid should be assigned to each machine. Notice also, that the benefit gained from using dynamic performance forecasts is substantial. Less obvious, however, is the improvement gained through resource selection. While the version that used resource selection does run between 25% and 50% faster than the non-selecting runtime AppLeS, the relative improvement compared to the blocked implementation is not large. However, the range of feasible partitions for the non-selecting runtime AppLeS is limited. For example, under the conditions during which the experiments were conducted, it was not possible to balance the execution time for a 500 by 500 element problem: the communication delay between UCSD and SDSC was so great that processors in either end would need to compute for a negative amount of time to compensate.

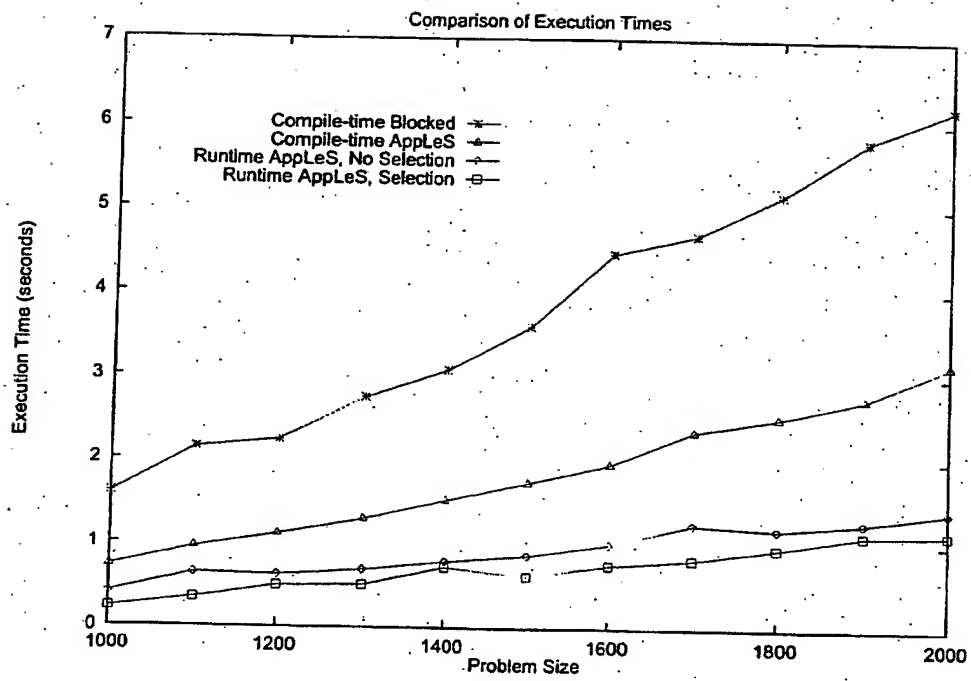


Figure 6: Execution times for Jacobi2D.

✓

In Figure 7 we show execution time data for a wider range of problem sizes using Compile-time Blocked, and the full AppLeS partitioner. Without resource selection, AppLeS would only be able to compute reliably (depending on contention conditions) over the 1000 to 2000 problem size domain. We also show the predicted execution time AppLeS computed for each run. For each problem size, we plot the time that the performance model predicted against the actual execution time that resulted for each mapping. It is the accuracy of the performance model that allows AppLeS to choose good resource mappings.

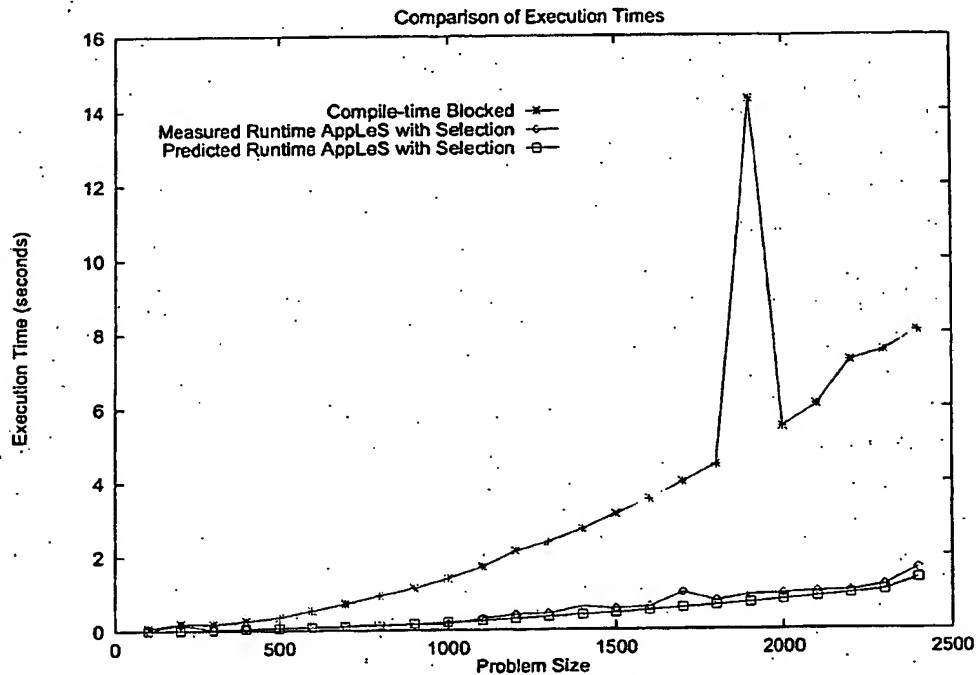


Figure 7: Execution times for Jacobi2D.

Note also the large spike in execution time for the blocked partitioning at the 1900 problem size abscissa. During one experimental run at that size, a network gateway between UCSD and SDSC went down forcing all communications between the two to use an alternative and much slower route. The AppLeS agent (through Network Weather Service readings) was able to detect the sudden drop in available bandwidth and avoid partitionings that spanned the affected link.

3.2 Partitioning for Memory Availability

Distributed parallel execution also allows an application to aggregate memory resources so that problems that are larger than will fit into any single memory may be solved. Indeed,

the motivation behind the parallel implementation of many codes stems from the need to use collections of memory systems rather than a desire for concurrent execution. To investigate the ability of the AppLeS approach to effectively aggregate memory, we added to the resource pool two IBM SP-2 processors with 128 megabytes of real memory each. The SP-2 uses virtual memory on each of its nodes so that more than 128 megabytes of memory may be used. However, memory is paged to disk causing reference times to increase dramatically when the real memory of the system is exceeded. During the experiments, we had dedicated access to the two SP-2 processors and the link between them, but they were connected to the rest of the resources via a shared ethernet segment. Figure 8 shows the resource pool including the SP-2 nodes:

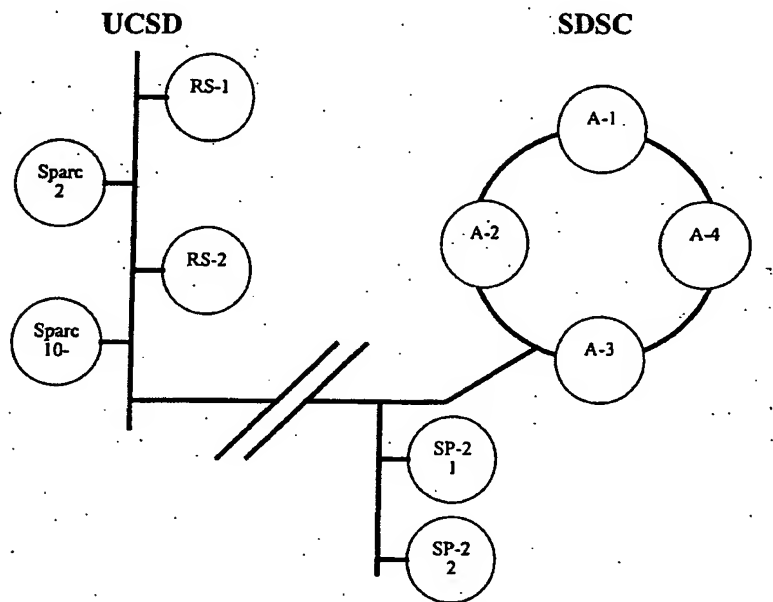


Figure 8: Resource Pool Including SP-2 Processors.

Since the processors were completely unloaded, and their connectivity to the other resources suffered from contention, the best partitioning (yielding the shortest execution time) was to split the grid evenly between the two SP-2 nodes as long as neither partition exceeded the available real memory on each node. However, when the problem size caused the partitions to spill out of the available real memory, the resulting delays due to paging caused execution time to increase substantially. In Figure 9 we show the execution time of a blocked partitioning using the SP-2 processors only versus the AppLeS approach for Jacobi2D.

For problem sizes less than 3900 by 3900, AppLeS correctly chose the mapping using the SP-2 processors and exhibited nearly identical execution times to the blocked mapping. As problem size increased, the SP-2 began paging, causing execution time to increase to the

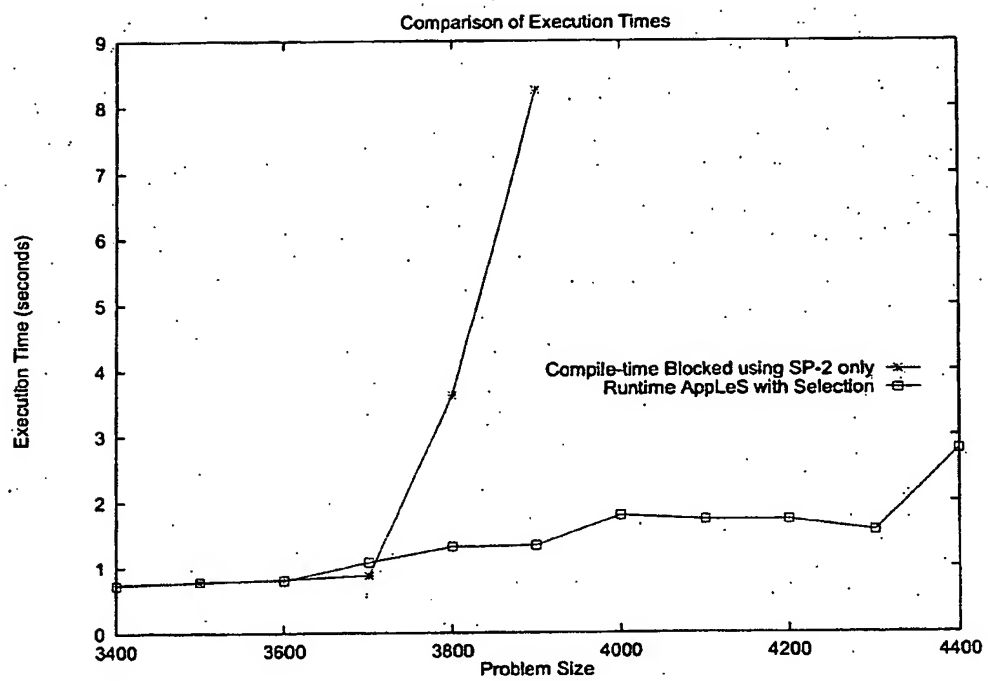


Figure 9: Partitioning and Memory Usage.

point where use of these processors was no longer feasible. The AppLeS agent was able to locate memory elsewhere within the resource pool effectively. At each problem size beyond 3900, the AppLeS was able to find memory it could use effectively without a dramatic change in the performance trajectory.

Thus far we have shown how the AppLeS approach was used effectively to determine a performance-efficient (and non-obvious) schedule for Jacobi2D. It was important to walk through this example in detail to demonstrate this approach. We now discuss how the AppLeS approach can be used as the basis for the design of general software agents which facilitate application-level scheduling for distributed parallel applications.

4 Developing General AppLeS Agents

It is clear from the previous sections that application-level scheduling can be used effectively to achieve performance for distributed applications. However, to develop general AppLeS agents, we must convince ourselves that the following questions could be answered in the affirmative:

- Is the application-level approach for selecting a performance-efficient schedule generalizable?
- Is it possible to efficiently obtain the appropriate level of application and system information (from the user or through analysis) from which good schedules may be derived?

To address the first question, observe that in the development of the application-level schedule for Jacobi2D, our approach did not rely particularly on the choice of algorithm, implementation language, or programming style for success. The organization of the AppLeS software mimics how a diligent user would schedule his or her application. The characteristics of the application are relevant only as they pertain to modeling its performance. In AppLeS, we modularize application-specific, system-specific and dynamic information and use this information to parameterize the general approach.

To address the second question, we developed a set of data sources to provide the relevant application- and system-specific information efficiently. The Network Weather Service was designed to provide dynamic system information and short-term forecasts. Application-specific information is provided through a Heterogeneous Application Template (or HAT) which distills much of the information from the application relevant to performance estimation. Additional information which reflects the user's preferences, access to resources, etc. is provided by User Specifications. Note that for AppLeS, as in practice, the more complete the application information that is available to the scheduler, the better the schedule.

AppLeS is currently a work-in-progress. The software has been designed and the underlying building blocks are currently being prototyped. We are working with researchers from the Legion project [12], [17] and from the Globus project [3], [11] to prototype AppLeS as an application-level scheduler for these resource management systems. In addition, we are progressing on an implementation which uses MPI as the underlying substrate.

Note that AppLeS essentially develops a customized scheduler for each application. This differs from the approach taken in much of the scheduling literature ([21], [13], [19], [23] [7]

etc.). Application-level scheduling is related to the work of Brewer [2], and more directly to the Mars project [8]. Brewer's work, which attempts to select the correct implementation of an algorithm for a given machine based on a small set of static parameters, uses application-specific information to improve performance. The MARS project [8], whose goal is to produce more general-purpose software, is more similar in scope and intent to AppLeS. An important difference, however, is that AppLeS includes user-specific as well as application-specific information in its scheduling decisions. User-specific information provides a powerful and well-defined interface that allows the user to influence and control how the scheduling agent will behave.

In the following sections, we describe the architecture for general AppLeS agents.

4.1 The AppLeS Organization

AppLeS is organized in terms of four subsystems and a single active agent called the **Coordinator**. The four subsystems are

- **The Resource Selector** which chooses and filters different resource combinations for the application's execution,
- **The Planner** which generates a description of a system-independent schedule from a resource combination,
- **The Performance Estimator** which generates a performance estimate for candidate schedules according to the user's performance metric, and
- **The Actuator** which implements the "best" schedule on the target resource management system(s).

Figure 10 depicts the Coordinator and these four subsystems. Application-specific, system-specific, and dynamic information used by these subsystems constitute an "information pool" which all subsystems share. There are four general sources of information feeding the information pool. The **Network Weather Service** provides dynamic information on system state and forecasts of system state for the time frame in which the application will be scheduled. The **Heterogeneous Application Template** is a web-oriented interface in which the user provides specific information about the structure, characteristics and current implementations of the application and its tasks. The **User Specifications** provide information on the user's criteria for performance, preferences for implementation, additional application information, etc. Finally, the **Model pool** provides model templates used by the AppLeS subsystems for application performance estimation.

AppLeS agents will be employed as follows: Initially, the user provides information to the agent via the HAT and User Specifications. The agent uses the Resource Selector to select a set of viable resource configurations based on accessibility, the user's access rights, the characteristics of the application (input as filters which exclude resources that are not viable), and a notion of "distance" which is derived from HAT information and the Model pool, or provided as a default by the Coordinator. For each viable resource configuration, the Planner (in conjunction with the Performance Estimator and the Network Weather Service)

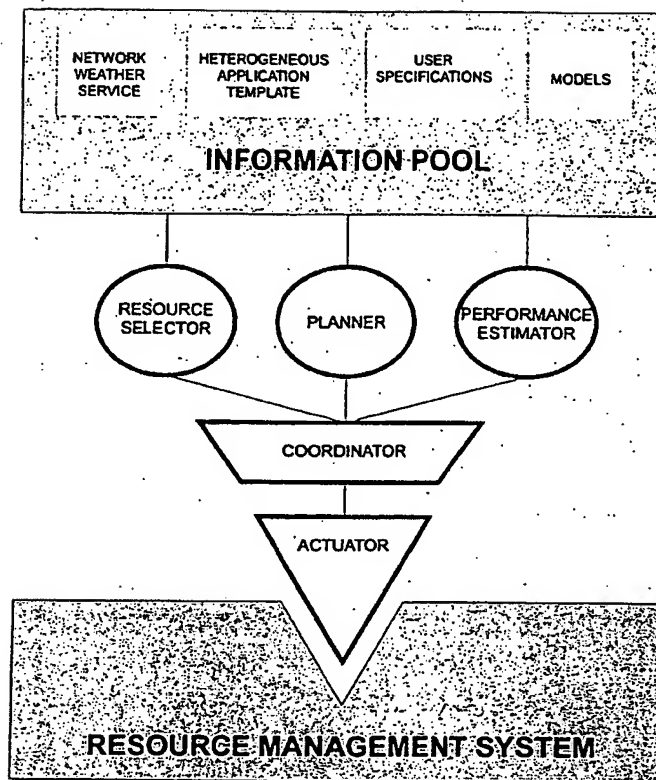


Figure 10: Relationship of the components of AppLeS.

computes a potential schedule of the resources using predictive models from the Model pool. The Coordinator considers the performance of the candidate schedules and selects a “best” schedule for implementation. The Actuator then interacts with the resource management system(s) to implement this schedule.

In the following subsections, we describe each of the components of AppLeS agents in more detail.

4.2 The Coordinator

The Coordinator embodies the active thread or threads of control within an AppLeS agent. It executes a **blueprint** that dictates the way in which it uses the various other subsystems to derive a schedule, initiate the application, and monitor its progress. The blueprint can be specified by the user or by the system for a particular application or class of applications (e.g. data parallel applications). We show a sample blueprint in Figure 11. This is typical for a user scheduling a minimum execution time application over a large set of possible resources,

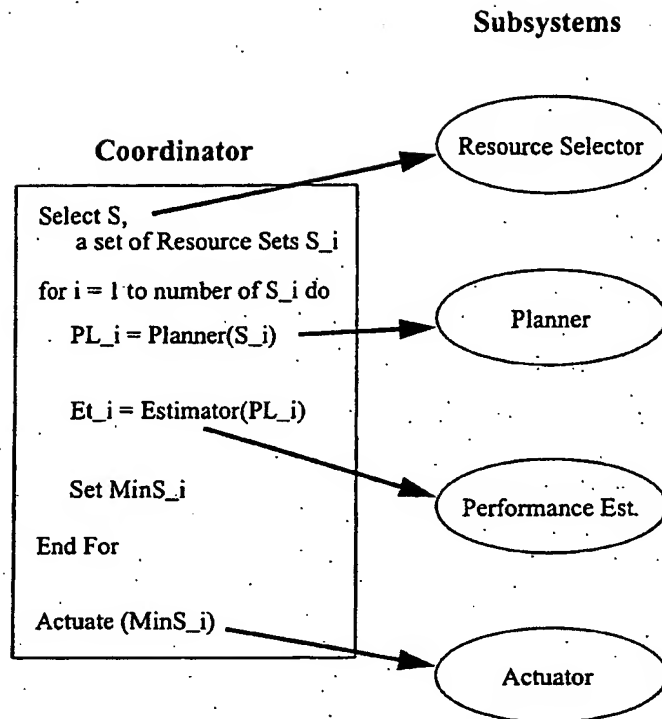


Figure 11: Coordinator and Blueprint

and is, in fact, the blueprint used to schedule Jacobi2D.

4.3 Data Sources

While the Coordinator directs the interactions between subsystems through its blueprint, each subsystem draws upon a variety of data sources to perform its function. These data sources contribute information to a data pool which is available to all AppLeS functions (see Figure 10). They are the Network Weather Service (NWS), the Heterogeneous Application Template (HAT), the User Specifications, and the Model pool. In this section, we briefly describe the form and content of each.

4.3.1 The Network Weather Service

The Network Weather Service provides software for monitoring and predicting the load (or "weather") on networked resources. Our strategy is to use sensors to dynamically probe and read the network "weather" conditions such as CPU load, available free memory, network performance, etc.

To provide forecasts of system state, the Network Weather Service uses a number of stochastic techniques for predicting network load. Experiments using different network links and predictors show that, in general, for a given resource, different estimation techniques will yield the best forecasts at different times. Consequently, the Network Weather Service tracks the error between all predictors and sampled data, and uses the predictor with the lowest cumulative error to make predictions of system state. Both the prediction and a measure of its recent "accuracy" are used by the Resource Selector, the Planner, and the Performance Estimator subsystems of an AppLeS agent.

We have prototyped this facility with good results as shown in the Jacobi2D example. We are currently integrating Network Weather Service facilities with the Legion and Globus resource management systems.

4.3.2 The Heterogeneous Application Template

The Heterogeneous Application Template (HAT) provides basic information about the overall application, tasks and implementations in terms of their resource requirements. Information is provided through a web interface which makes explicit the structural parameters of the application, information about existing implementations of application tasks, and the data movement requirements between distinct tasks. Figures 12, 13 and 14 give a sample of HAT parameters.

The HAT also lets the user identify an active set, i.e. a set of task/machine implementations that work together to compose an entire application. Since there may be multiple implementations, the active set identifies the particular task/machine allocations that will be used in a single full implementation of the application. For Jacobi2D, the active set was composed of a single task implementation per machine. In general, however, there may be several implementations from which to choose and multiple active sets.

Notice that the user may not have all the information requested by HAT. The system can use partial information to determine a schedule. However, as is the case for the user, the better and more comprehensive the information available, the more performance-efficient the schedule is likely to be.

4.3.3 User Specifications

While the HAT describes application-specific information, information specific to a particular user or application developer is made available to an AppLeS through User Specifications (US) which will also be html-based. The most important role of the US is in the definition of user-specified requirements which fall into the three broad categories: execution constraints, performance objectives, and user preferences. Execution constraints refer to the access rights and resource constraints of the user. The user's performance objective is also conveyed through the US. For Jacobi2D, minimum execution time was the desired objective. Finally, the US allows the user to specify preferences for the Coordinator to attempt to satisfy. It may be that one resource should be preferred over another for non-performance-related reasons. This feature gives the user tremendous control over the actions of AppLeS and the solutions that it generates.

4.3.4 Models

The Model pool provides a set of model templates which are used for application performance estimation by the Planner, Performance Estimator and Resource Selector. Model templates are structures for composing models of characteristics which contribute to application performance. For example, in Jacobi, the model template for the execution time for processor i is

$$T_i = \text{Computation} + \text{Communication}$$

where Computation is instantiated as $A_i * P_i$ and Communication is instantiated as C_i , as described in Section 2.1.

Model templates may be provided by the user. Default model templates for classes of applications (e.g. data parallel regular grid applications) will be available in the Model pool. Note that model templates can leverage successful models from the literature such as [2], [18] [10], [24], [22], etc. to predict the performance of the application and its tasks.

4.4 Resource Selector

The Resource Selector produces viable active sets to be considered by the Coordinator. It may iterate multiple times to identify a set of candidate active sets according to different selection criteria.

A potentially viable active set may be filtered to ensure its feasibility. Resources are prioritized with respect to an application-specific valuation function such as "distance", and filters are applied to the resource set to eliminate resources that are unusable. A filter may use information such as the user's access rights, memory constraints, implementation availability, etc. to eliminate resources quickly. Viable and feasible resource configurations will be scheduled by the Planner, evaluated by the Performance Estimator, and compared by the Coordinator to other candidate schedules.

In the Jacobi2D example, filters considered two characteristics of each potential schedule: the area of region i , A_i , and the available memory. Partitions with strips in which A_i was negative were filtered out. Next, resources which did not meet the memory requirements of application tasks were also filtered out. Such constraints for most users are readily identifiable, and can be used profitably to reduce the resource selection space.

4.5 Planner

The function of the Planner is to create a schedule for a feasible active set. The schedule is based on a scheduling policy that optimizes for the user's performance measure. In practice, most users will employ common performance measures (execution time, cost, speedup), and the Planner will be equipped with default scheduling policies for these measures if the user chooses not to recommend a policy of his/her own. The schedule generated by the Planner must be in a format that the Actuator (described in section 4.7) can implement on the target resource system.

In the Jacobi2D example, the Planner implemented a time-balancing scheduling policy. It took a list of candidate machines and their communication links (the feasible resource

set), and produced a mapping of grid strips to the machines. The Coordinator then used the Performance Estimator to determine the execution time of each mapping generated by the Planner and passed the best schedule to the Actuator.

4.6 The Performance Estimator

The performance estimator parameterizes a model template with component models to produce an estimate of application performance, given a schedule provided by the Planner. Parameters for the component models can be provided by the user or derived from other data sources such as the Network Weather Service. Since dynamic information is included, the resulting estimates can be targeted to the time frame during which the application will be run by the Actuator. In Jacobi, the formula $T_i = A_i * P_i + C_i$ is evaluated to obtain an estimate of the time necessary to compute each strip.

Note that it is important to estimate the behavior of the application tasks in the context of the production systems in which they will be used. For this reason, we are developing models which forecast the slowdown of tasks on shared resources (networks and machines) [4]. Factoring slowdown into the model will provide a more realistic estimate of application and task performance in the presence of contention.

4.7 Actuator

AppLeS does not function as a resource manager – it relies on the services of existing resource managers to perform resource allocation and task instantiation. It is the job of the Actuator to implement the schedule (determined by the Planner) using the semantics and facilities supported by the target resource management system. Some of these resource managers, such as PVM, are limited in scope and provide little additional functionality. Others, like Legion, have the potential for communicating considerable information about resource and application status. The Actuator will also convey whatever feedback information is available to the various subsystems. It acts as the conduit between the Coordinator and the underlying resource management facilities.

The minimum functionality required by the Actuator is the ability to initiate a network connected task on a remote machine. More accurate scheduling can be accomplished when the resource management system returns feedback about when resources are actually available for use, or can provide guaranteed service times in response to requests for service. Since the AppLeS agent is working at the application level, however, the Actuator minimally has access to whatever facilities the application enjoys. It will use the same facilities to communicate with the application and manage its task execution that the application itself uses to control its tasks. In that sense, the Actuator, and by extension the AppLeS agent, constitute an integrated extension of the program being scheduled. AppLeS and the application become part of the same execution instance. In the Jacobi2D example, the Actuator issued KeLP directives to control grid partitioning. These were the same primitives the application used to manage the grid itself.

5 Summary

As network speeds increase and parallel distributed computing becomes more prevalent, resource-intensive applications will increasingly need to leverage shared, heterogeneous networks of resources. Effective coordination of application components and their use of resources is key to performance. In this work, we described **application-level scheduling** as a way of achieving performance-efficient schedules for applications which execute on heterogeneous networks of machines. We described principles which reflect the way in which applications are scheduled by their end-users and illustrated these principles by developing a "proof-of-concept" application-level scheduler for a distributed data-parallel Jacobi application. We then described a general architecture for Application-Level Schedulers and described the subsystems which compose an AppLeS agent.

From the results generated by our prototype, it is clear that the AppLeS approach can achieve substantial performance improvements for an individual application over conventional scheduling methods. Application-level scheduling allows the user to deal with the heterogeneous system as it really is: under the control of multiple system schedulers, shared by other contending applications, and able to deliver only a dynamically varying fraction of resource performance. When such characteristics are explicitly factored into the scheduling activity, the application can better leverage the system to achieve performance.

Acknowledgments

We are grateful to the researchers in the UCSD Parallel Computation Laboratory, and in particular to Stephen J. Fink for many substantive discussions. We are also grateful to Andrew Grimshaw, Carl Kesselman, Reagan Moore, John Karpovich, Doug Shea, and Darren Atkinson for their thoughtful comments and support.

AppLeS Home Page

<http://www-cse.ucsd.edu/users/berman/apples.html>

References

- [1] Berman, F. and Moore, R., **Heterogeneous working group report**, Proceedings of the Second Pasadena Workshop on System Software and Tools for High Performance Computing Environments, 1995. <http://cesdis.gsfc.nasa.gov/PAS2.index.html>.
- [2] Brewer, E. A., **High-level optimization via automated statistical modeling**, Proceedings of Principles and Practice of Parallel Programming, PPOPP'95 (1995), pp. 80-91.

- [3] DeFanti, T., Foster, I., Papka, M., Stevens, R. and Kuhfuss, T., **Overview of the I-way: Wide area visual supercomputing**, to appear in the International Journal of Supercomputer Applications.
- [4] Figueira, S. M. and Berman, F., **Modeling the effects of contention on the performance of heterogeneous applications**, to appear in the Proceedings of the High Performance Distributed Computing Conference (1996).
- [5] Fink, S., <http://www-cse.ucsd.edu/groups/hpcl/scg/kelp.html>, 1995.
- [6] Fink, S. J., Baden, S. B. and Kohn, S. R., **Flexible communication mechanisms for dynamic structured applications**, in preparation, 1996.
- [7] Freund, R., Proceedings of the 1996 IPPS Workshop on Heterogeneous Computing.
- [8] Gehrmf, J. and Reinfeld, A., **Mars - a framework for minimizing the job execution time in a metacomputing environment**, Proceedings of Future general Computer Systems (1996).
- [9] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V., **PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing**, MIT Press, 1994.
- [10] Getov, V. S., Hockney, R. W. and Hey, A. J. G., **Performance analysis of distributed applications by suitability functions**, in Proceedings of the MPPM Conference (1993).
- [11] Globus, <http://www.mcs.anl.gov/globus>.
- [12] Grimshaw, A. S., Wulf, W. A., French, J. C., Weaver, A. C. and Reynolds, P. F., **Legion: The next logical step toward a nationwide virtual computer**, Tech. Rep. CS-94-21, University of Virginia, 1994.
- [13] Hensgen, D. A., Moore, L., Kidd, T., Freund, R., Keith, E., Kussow, M., Lima, J. and Campbell, M., **Adding rescheduling to and integrating condor with smartnet**, in Proceedings of the Heterogeneous workshop (1995).
- [14] High Performance Fortran Forum, **High performance fortran language specification**, Rice University, Houston, Texas, May 1993.
- [15] Hoffman, J. D., **Numerical Methods for Engineers and Scientists**, McGraw-Hill, Inc, 1992.
- [16] Korab, H. and Brown, M., **Virtual environments and distributed computing at SC'95: GII testbed and HPC challenge applications on the I-way**, in Proceedings of Supercomputing '95 (1995).
- [17] Legion, <http://www.cs.virginia.edu/~mentat/legion/legion.html>.

- [18] Messina, P. and Heirich, A., personal communication, 1995.
- [19] Pruyne, J. and Livny, M., **Parallel processing on dynamic resources with Carmi**, in Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '95 (April 1995).
- [20] PVM, <http://www.epm.ornl.gov:80/pvm/>.
- [21] Rudolph, L. and Feitelson, D., Proceedings of the 1996 IPPS Workshop on Job Scheduling Strategies for Parallel Processing (1996).
- [22] Sarkar, V., **Automatic partitioning of a program dependence graph into parallel tasks**, IBM Journal of Research and Development 35, 5/6 (Sept/Nov 1991).
- [23] Siegel, H., Antonio, J., Metzger, R., Tan, M. and Li, Y. A., **Heterogeneous computing**. Tech. Rep., Purdue University EE Technical Report TR-EE-94-37.
- [24] Zhang, X. and Yan, Y., **A framework of performance prediction of parallel computing nondedicated heterogeneous now**, in Proceedings of the 1995 International Conference on Parallel Processing (1995), pp. 163-7.

HAT - Heterogeneous Application Template

APPLICATION:

USER:

[User Account Information](#) [User Specification Information](#)

HAT - Structure Template

INPUT:
 Amount of data needed to start application
 MBytes
 Current source (give full machine name e.g. paragate.sdsc.edu)

OUTPUT:
 Amount of data returned by application
 MBytes
 Current source (give full machine name e.g. paragate.sdsc.edu)

ITERATION PHASE: [Create new iteration phase](#)

[Listing of Implementations](#)
[Listing of Active Sets](#)

[Structure](#) [Implementation](#) [Interface](#) [Help](#) [AppLeS Manager](#)

Figure 12: The Structure module of HAT gives information about the general functional decomposition of the application, and lets a user identify an active set for the application.

HAT - Implementation Template	
TASK:	
PLATFORM:	
PARADIGM:	
<input type="checkbox"/> Sequential <input type="checkbox"/> Vector <input type="checkbox"/> Task Parallel <input type="checkbox"/> Data Parallel	
<input type="checkbox"/> Single Processor <input type="checkbox"/> Multi-Processor	
USAGE:	
<input type="checkbox"/> Dedicated <input type="checkbox"/> Non-dedicated	
DATA STRUCTURES:	
Number	<input type="text"/>
Size	<input type="text"/> Bytes
Computation per data structure	<input type="text"/> MFlops
Communication per data structure	<input type="text"/> Words
RATIO:	
<i>Select an approximation or fill in numerical values</i>	
<input type="checkbox"/> Communication Heavy <input type="checkbox"/> Balanced <input type="checkbox"/> Computation Heavy	
Computation per data structure	<input type="text"/> MFlops
Communication per data structure	<input type="text"/> Words
COMMUNICATION PATTERNS:	
<input type="checkbox"/> Pt to Pt <input type="checkbox"/> Stencil <input type="checkbox"/> Multicast <input type="checkbox"/> Broadcast	
MEMORY:	
Core memory needed for in-core sol'n	<input type="text"/> MWords
TUNING FACTOR:	
<input type="checkbox"/> 1 (bubblesort) <input type="checkbox"/> 3 (cs 101) <input type="checkbox"/> 5 (1st year grad) <input type="checkbox"/> 7 (PhD thesis) <input type="checkbox"/> 10 (hand tuned assembler)	
Structure	Implementation
Interface	Help
AppLeS Manager	

Figure 13: The Implementation module focuses on how the task was implemented for a specific platform.

HAT - Interface Template	
IMPLEMENTATION A:	
IMPLEMENTATION B:	
NETWORK: <input type="checkbox"/> Ethernet <input type="checkbox"/> Hippi <input type="checkbox"/> ATM <input type="checkbox"/> Other	
COMMUNICATION FREQUENCY: Per application iteration: <input type="text"/> MBytes	
AMMOUNT OF COMMUNICATION: Total: <input type="text"/> MBytes <input type="checkbox"/> Dependent on no. of iterations Per message: <input type="text"/> MBytes	
DATA CONVERSION: Conversion type: <input type="checkbox"/> Format Conversion <input type="checkbox"/> Structure Conversion Performed on: <input type="text"/>	
PIPELINE: <input type="checkbox"/> Pipelined Data <input type="checkbox"/> Strict Data Size of Pipeline: <input type="text"/> MBytes	
Structure Implementation Interface Help AppLeS Manager	

Figure 14: The Interface module of HAT characterizes the communication between implementations A and B mapped to distinct execution sites.



Redbooks Paper

Viktors Berstis

Fundamentals of Grid Computing

The purpose of this IBM Redpaper is to provide discussion material about grid computing, concepts, use, and architecture. Grid computing represents unlimited opportunities in terms of business and technical aspects. The audience for this paper are all hungry minds looking for a collection of facts and data about this new and exciting realm.

The following major topics will be introduced to the readers:

- ▶ What grid computing can do
- ▶ Grid concepts and components
- ▶ Grid construction
- ▶ The present and the future
- ▶ What the grid cannot do

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources.

The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing.

What grid computing can do

When you deploy a grid, it will be to meet a set of customer requirements. To better match grid computing capabilities to those requirements, it is useful to keep in mind the reasons for using grid computing. This section describes the most important capabilities of grid computing.

Exploiting underutilized resources

The easiest use of grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the grid.

There are at least two prerequisites for this scenario. First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application.

For example, a batch job that spends a significant amount of time processing a set of input data to produce an output set is perhaps the most ideal and simple use for a grid. If the quantities of input and output are large, more thought and planning might be required to efficiently use the grid for such a job. It would usually not make sense to use a word processor remotely on a grid because there would probably be greater delays and more potential points of failure.

In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5% of the time. In some organizations, even the server machines can often be relatively idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage.

The processing resources are not the only ones that may be underutilized. Often, machines may have enormous unused disk drive capacity. Grid computing, more specifically, a "data grid", can be used to aggregate this unused storage into a much larger virtual data store, possibly configured to achieve improved performance and reliability over that of any single machine.

If a batch job needs to read a large amount of data, this data could be automatically replicated at various strategic points in the grid. Thus, if the job must be executed on a remote machine in the grid, the data is already there and does not need to be moved to that remote point. This offers clear performance benefits. Also, such copies of data can be used as backups when the primary copies are damaged or unavailable.

Another function of the grid is to better balance resource utilization. An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid enabled, they can be moved to underutilized machines during such peaks. In fact, some grid implementations can migrate partially completed jobs. In general, a grid can provide a consistent way to balance the loads on a wider federation of resources. This applies to CPU, storage, and many other kinds of resources that may be available on a grid. Management can use a grid to better view the usage patterns in the larger organization, permitting better planning when upgrading systems, increasing capacity, or retiring computing resources no longer needed.

Parallel CPU capacity

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others.

The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts. A CPU intensive grid application can be thought of as many smaller "subjobs," each executing on a different machine in the grid. To the extent that these subjobs do not need to communicate with each other, the more "scalable" the application becomes. A perfectly scalable application will, for example, finish 10 times faster if it uses 10 times the number of processors.

Barriers often exist to perfect scalability. The first barrier depends on the algorithms used for splitting the application among many CPUs. If the algorithm can only be split into a limited number of independently running parts, then that forms a scalability barrier. The second

barrier appears if the parts are not completely independent; this can cause contention, which can limit scalability. For example, if all of the subjobs need to read and write from one common file or database, the access limits of that file or database will become the limiting factor in the application's scalability. Other sources of inter-job contention in a parallel grid application include message communications latencies among the jobs, network communication capacities, synchronization protocols, input-output bandwidth to devices and storage devices, and latencies interfering with real-time requirements.

Applications

There are many factors to consider in grid-enabling an application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability. Furthermore, there are no practical tools for transforming arbitrary applications to exploit the parallel capabilities of a grid. There are some practical tools that skilled application designers can use to write a parallel grid application. However, automatic transformation of applications is a science in its infancy. This can be a difficult job and often requires top mathematics and programming talents, if it is even possible in a given situation. New computation intensive applications written today are being designed for parallel execution and these will be easily grid enabled, if they do not already follow emerging grid protocols and standards.

Virtual resources and virtual organizations for collaboration

Another important grid computing contribution is to enable and simplify collaboration among a wider audience. In the past, distributed computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources, as illustrated in Figure 1 on page 4. The users of the grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. These virtual organizations can share their resources collectively as a larger grid.

Sharing starts with data in the form of files or databases. A "data grid" can expand data capabilities in several ways. First, files or databases can seamlessly span many systems and thus have larger capacities than on any single system. Such spanning can improve data transfer rates through the use of striping techniques. Data can be duplicated throughout the grid to serve as a backup and can be hosted on or near the machines most likely to need the data, in conjunction with advanced scheduling techniques.

Sharing is not limited to files, but also includes many other resources, such as equipment, software, services, licenses, and others. These resources are "virtualized" to give them a more uniform interoperability among heterogeneous grid participants.

The participants and users of the grid can be members of several real and virtual organizations. The grid can help in enforcing security rules among them and implement policies, which can resolve priorities for both resources and users.

BEST AVAILABLE COPY

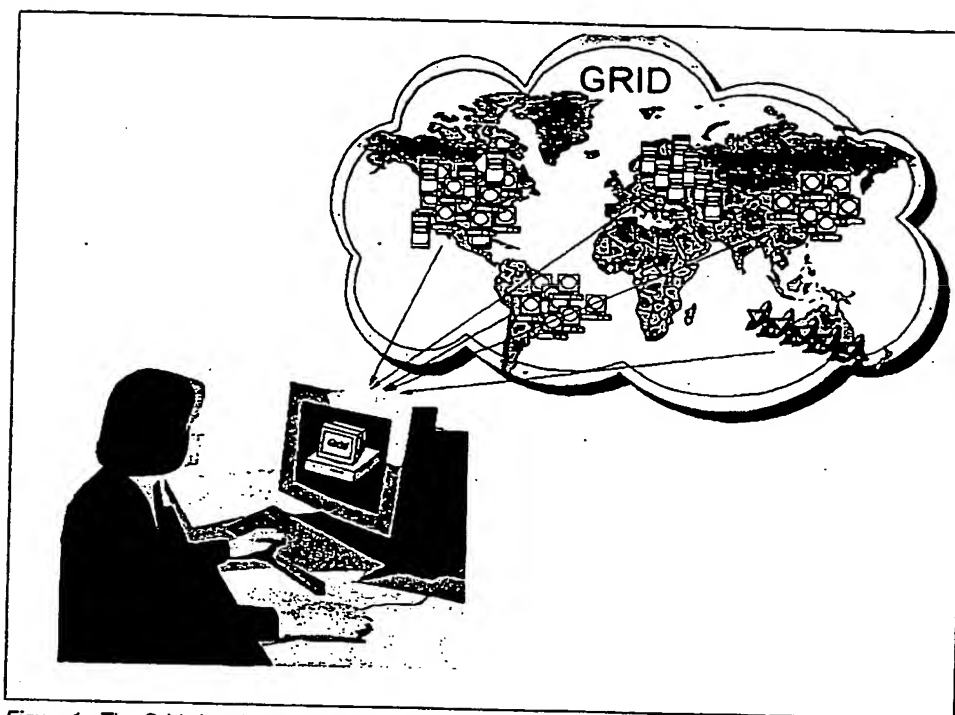


Figure 1 The Grid virtualizes heterogeneous and geographically disperse resources for each virtual organization presenting a simpler view

Access to additional resources

In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity.

For example, if a user needs to increase his total bandwidth to the Internet to implement a data mining search engine, the work can be split among grid machines that have independent connections to the Internet. In this way, the total searching capability is multiplied, since each machine has a separate connection to the Internet. If the machines had shared the connection to the Internet, there would not have been an effective increase in bandwidth.

Some machines may have expensive licensed software installed that the user requires. His jobs can be sent to such machines more fully exploiting the software licenses.

Some machines on the grid may have special devices. Most of us have used remote printers, perhaps with advanced color capabilities or faster speeds. Similarly, a grid can be used to make use of other special equipment. For example, a machine may have a high speed, self feeding, DVD writer that could be used to publish a quantity of data faster. Some machines on the grid may be connected to scanning electron microscopes that can be operated remotely. In this case, scheduling and reservation are important. A specimen could be sent in advance to the facility hosting the microscope. Then the user can remotely operate the machine, changing perspective views until the desired image is captured.

The grid can enable more elaborate access, potentially to remote medical diagnostic and robotic surgery tools with two-way interaction from a distance. The variations are limited only by one's imagination. Today, we have remote device drivers for printers. Eventually, we will

see standards for grid enabled device drivers to many unusual devices and resources. All of these will make the grid look like a large virtual machine with a collection of virtual resources beyond what would be available on just one conventional machine.

Resource balancing

A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization, as illustrated in Figure 2. This feature can prove invaluable for handling occasional peak loads of activity in parts of an larger organization. This can happen in two ways:

- An unexpected peak can be routed to relatively idle machines in the grid.
- If the grid is already fully utilized, the lowest priority work being performed on the grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work.

Without a grid infrastructure, such balancing decisions are difficult to prioritize and execute.

Occasionally, a project may suddenly rise in importance with a specific deadline. A grid cannot perform a miracle and achieve a deadline when it is already too close. However, if the size of the job is known, if it is a kind of job that can be sufficiently split into subjobs, and if enough resources are available after preempting lower priority work, a grid can bring a very large amount of processing power to solve the problem. In such situations, a grid can, with some planning, succeed in meeting a surprise deadline.

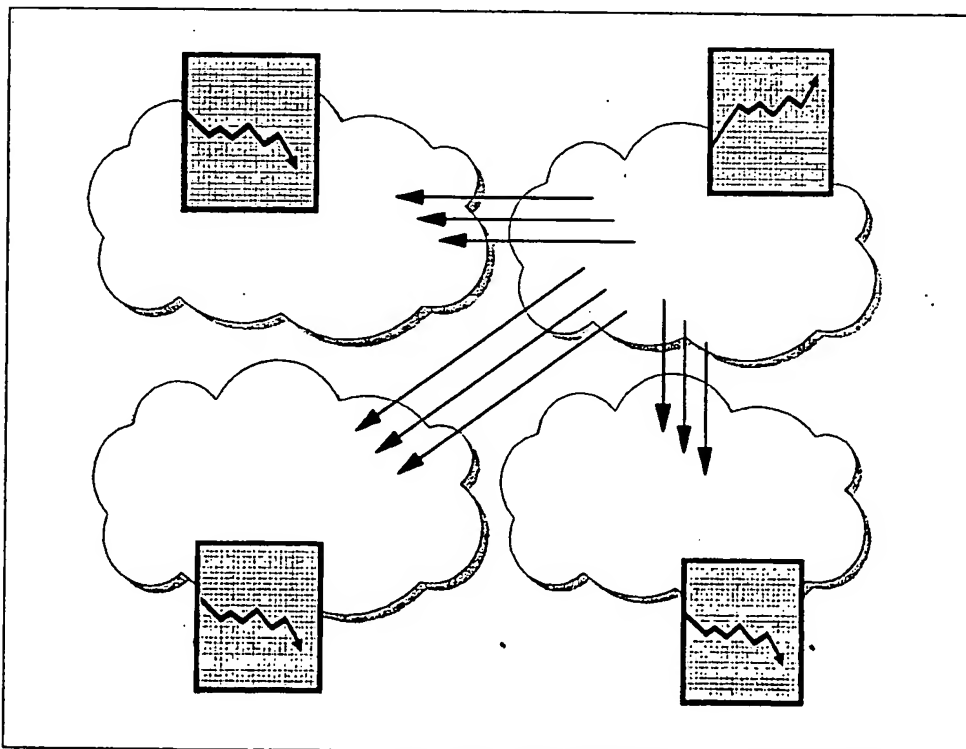


Figure 2 Jobs are migrated to less busy parts of the grid to balance resource loads and absorb unexpected peaks of activity in a part of an organization

Other more subtle benefits can occur using a grid for load balancing. When jobs communicate with each other, the Internet, or with storage resources, an advanced scheduler could schedule them to minimize communications traffic or minimize the distance of the communications. This can potentially reduce communication and other forms of contention in the grid.

Finally, a grid provides excellent infrastructure for brokering resources. Individual resources can be profiled to determine their availability and their capacity, and this can be factored into scheduling on the grid. Different organizations participating in the grid can build up grid credits and use them at times when they need additional resources. This can form the basis for grid accounting and the ability to more fairly distribute work on the grid.

Reliability

High-end conventional computing systems use expensive hardware to increase reliability. They are built using chips with redundant circuits that vote on results, and contain much logic to achieve graceful recovery from an assortment of hardware failures. The machines also use duplicate processors with hot pluggability so that when they fail, one can be replaced without turning the other off. Power supplies and cooling systems are duplicated. The systems are operated on special power sources that can start generators if utility power is interrupted. All of this builds a reliable system, but at a great cost, due to the duplication of high-reliability components.

In the future, we will see an alternate approach to reliability that relies more on software technology than expensive hardware. A grid is just the beginning of such technology. The systems in a grid can be relatively inexpensive and geographically dispersed. Thus, if there is a power or other kind of failure at one location, the other parts of the grid are not likely to be affected. Grid management software can automatically resubmit jobs to other machines on the grid when a failure is detected. In critical, real-time situations, multiple copies of the important jobs can be run on different machines throughout the grid, as illustrated in Figure 3 on page 7. Their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering.

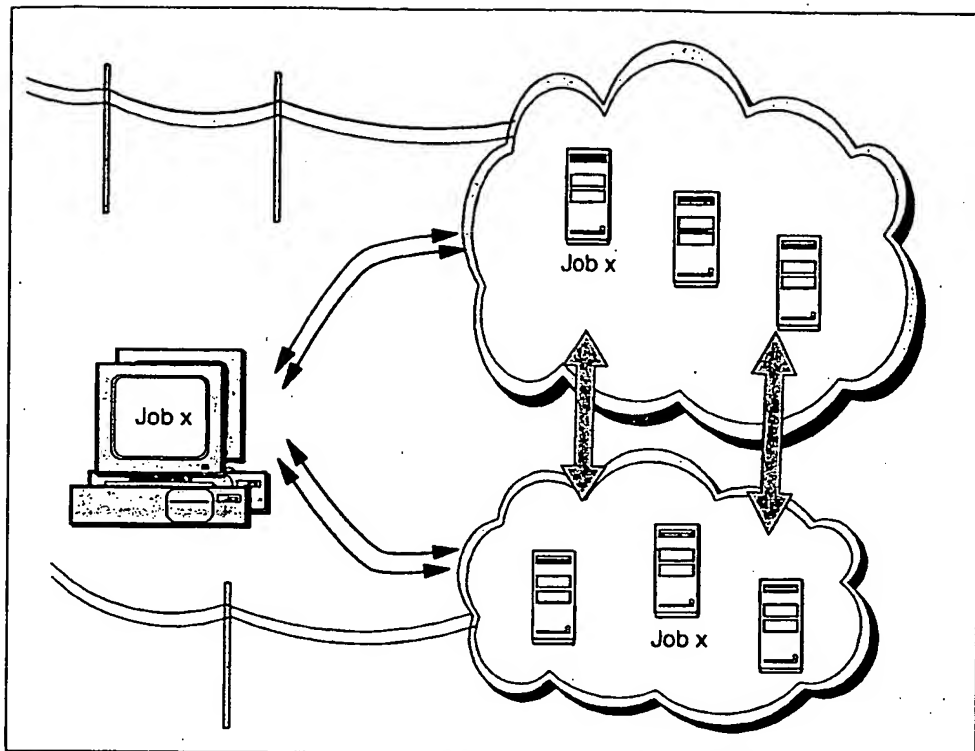


Figure 3 Redundant grid configuration and redundant job submission used to achieve high reliability

Such grid systems will utilize “autonomic computing.” This is a type of software that automatically heals problems in the grid, perhaps even before an operator or manager is aware of them. In principle, most of the reliability attributes achieved using hardware in today’s high availability systems can be achieved using software in a grid setting in the future.

Management

The goal to virtualize the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more dispersed IT infrastructure. It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization.

The grid offers management of priorities among different projects. In the past, each project may have been responsible for its own IT resource hardware and the expenses associated with it. Often this hardware might be underutilized while another project finds itself in trouble, needing more resources due to unexpected events. With the larger view a grid can offer, it becomes easier to control and manage such situations. As illustrated in Figure 4 on page 8, administrators can change any number of policies that affect how the different organizations might share or compete for resources.

Aggregating utilization data over a larger set of projects can enhance an organization’s ability to project future upgrade needs. When maintenance is required, grid work can be rerouted to other machines without crippling the projects involved.

Autonomic computing can come into play here too. Various tools may be able to identify important trends throughout the grid, informing management of those that require attention.

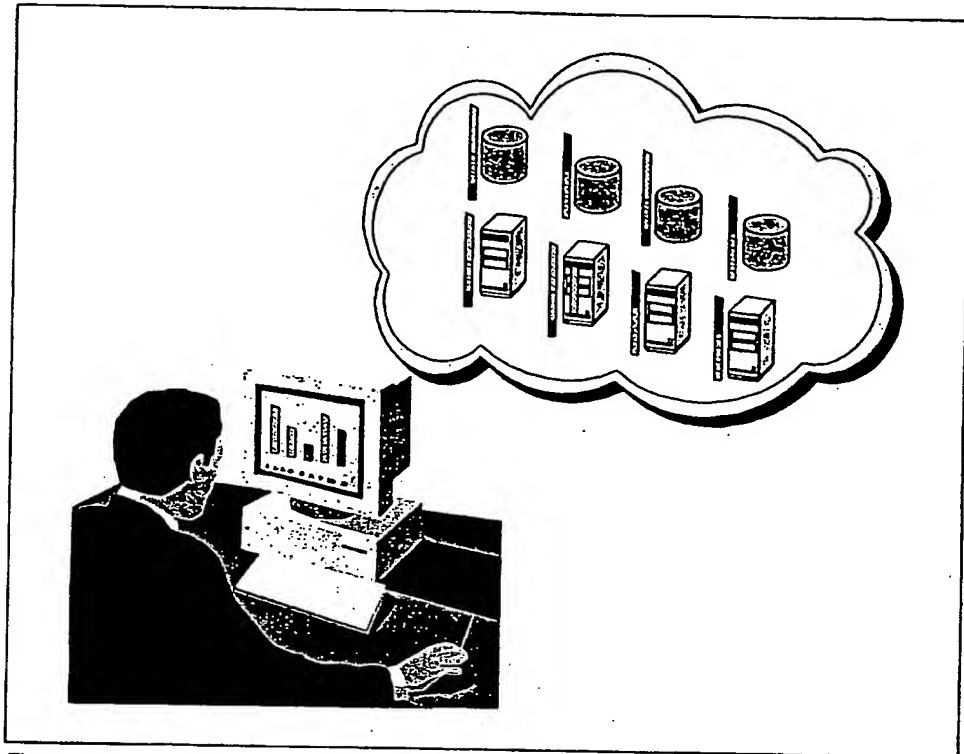


Figure 4 Administrators can adjust policies to better allocate resources

Grid concepts and components

In this section, we introduce the various grid concepts, components, and terms in more detail.

Types of resources

A grid is a collection of machines, sometimes referred to as "nodes," "resources," "members," "donors," "clients," "hosts," "engines," and many other such terms. They all contribute any combination of resources to the grid as a whole. Some resources may be used by all users of the grid while others may have specific restrictions.

Computation

The most common resource is computing cycles provided by the processors of the machines on the grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. There are three primary ways to exploit the computation resources of a grid. The first and simplest is to use it to run an existing application on an available machine on the grid rather than locally. The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors. The third is to run an application that needs to be executed many times on many different machines in the grid. "Scalability" is a measure of how efficiently the multiple processors on a grid are used. If twice as many processors makes an application complete in one half the time, then it is said to be perfectly scalable. However, there may be limits to scalability when applications can only be split into a limited number of

separately running parts or if those parts experience some other contention for resources of some kind.

Storage

The second most common resource used in a grid is data storage. A grid providing an integrated view of data storage is sometimes called a "data grid." Each machine on the grid usually provides some quantity of storage for grid use, even if temporary. Storage can be memory attached to the processor or it can be "secondary storage" using hard disk drives or other permanent storage media. Memory attached to a processor usually has very fast access but is volatile. It would best be used to cache data to serve as temporary storage for running applications.

Secondary storage in a grid can be used in interesting ways to increase capacity, performance, sharing, and reliability of data. Many grid systems use mountable networked file systems, such as Andrew File System (AFS), Network File System (NFS), Distributed File System (DFS), or General Parallel File System (GPFS). These offer varying degrees of performance, security features, and reliability features.

Capacity can be increased by using the storage on multiple machines with a unifying file system. Any individual file or data base can span several storage devices and machines, eliminating maximum size restrictions often imposed by file systems shipped with operating systems. A unifying file system can also provide a single uniform name space for grid storage. This makes it easier for users to reference data residing in the grid, without regard for its exact location. In a similar way, special data base software can "federate" an assortment of individual data bases and files to form a larger, more comprehensive data base, accessible using data base query functions.

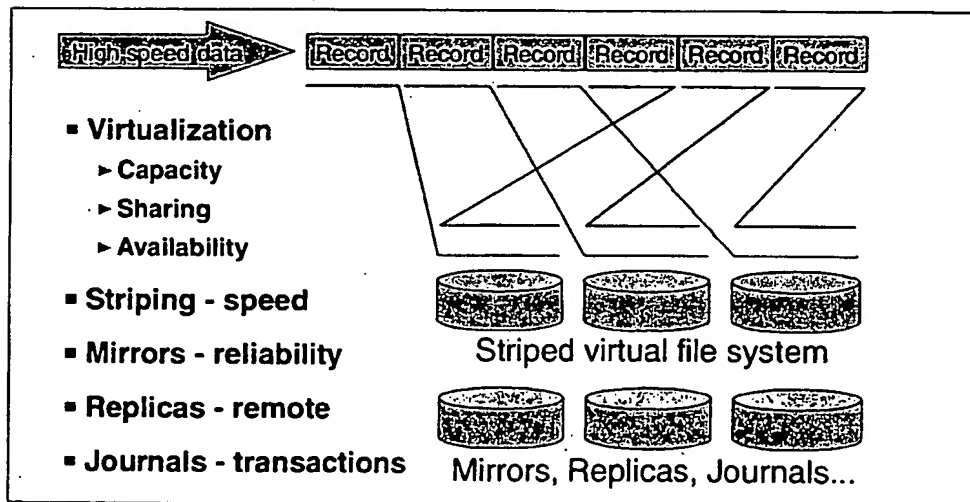


Figure 5 Data striping is writing or reading successive records to/from different physical devices, overlapping the access for faster throughput; additional techniques increase reliability

More advanced file systems on a grid can automatically duplicate sets of data, to provide redundancy for increased reliability and increased performance. An intelligent grid scheduler can help select the appropriate storage devices to hold data, based on usage patterns. Jobs can then be scheduled closer to the data, preferably on the machines directly connected to the storage devices holding the required data.

Data striping can also be implemented by grid file systems, as illustrated in Figure 5 on page 9. When there are sequential or predictable access patterns to data, this technique can create the virtual effect of having storage devices that can transfer data at a faster rate than any individual disk drive. This can be important for multimedia data streams or when collecting large quantities of data at extremely high rates from CAT scans or particle physics experiments, for example.

A grid file system can also implement journaling so that data can be recovered more reliably after certain kinds of failures. In addition, some file systems implement advanced synchronization mechanisms to reduce contention when data is shared and updated by many users.

Communications

The rapid growth in communication capacity among machines today makes grid computing practical, compared to the limited bandwidth available when distributed computing was first emerging. Therefore, it should not be a surprise that another important resource of a grid is data communication capacity. This includes communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of data to be processed and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid.

External communication access to the Internet, for example, can be valuable when building search engines. Machines on the grid may have connections to the external Internet in addition to the connectivity among the grid machines. When these connections do not share the same communication path, then they add to the total available bandwidth for accessing the Internet.

Redundant communication paths are sometimes needed to better handle potential network failures and excessive data traffic. In some cases, higher speed networks must be provided to meet the demands of jobs transferring larger amounts of data. A grid management system can better show the topology of the grid and highlight the communication bottlenecks. This information can in turn be used to plan for hardware upgrades.

Software and licenses

The grid may have software installed that may be too expensive to install on every grid machine. Using a grid, the jobs requiring this software are sent to the particular machines on which this software happens to be installed. When the licensing fees are significant, this approach can save significant expenses for an organization.

Some software licensing arrangements permit the software to be installed on all of the machines of a grid but may limit the number of installations that can be simultaneously used at any given instant. License management software keeps track of how many concurrent copies of the software are being used and prevents more than that number from executing at any given time. The grid job schedulers can be configured to take software licenses into account, optionally balancing them against other priorities or policies.

Special equipment, capacities, architectures, and policies

Platforms on the grid will often have different architectures, operating systems, devices, capacities, and equipment. Each of these items represents a different kind of resource that the grid can use as criteria for assigning jobs to machines. While some software may be available on several architectures, for example, PowerPC and x86, such software is often designed to run only on a particular type of hardware and operating system. Such attributes must be considered when assigning jobs to resources in the grid.

In some cases, the administrator of a grid may create a new artificial resource type that is used by schedulers to assign work according to policy rules or other constraints. For example, some machines may be designated to only be used for medical research. These would be identified as having a medical research attribute and the scheduler could be configured to only assign jobs that require machines of the medical research "resource." Others may participate in the grid only if they are not used for military purposes. In this situation, jobs requiring a "military resource" would not be assigned to such machines. Of course, the administrators would need to impose a classification on each kind of job through some certification procedure to use this kind of approach.

Jobs and applications

Although various kinds of resources on the grid may be shared and used, they are usually accessed via an executing "application" or "job." Usually we use the term "application" as the highest level of a piece of work on the grid. However, sometimes the term "job" is used equivalently. Applications may be broken down into any number of individual jobs, as illustrated in Figure 6. Those, in turn, can be further broken down into "subjobs." The grid industry uses other terms, such as transaction, work unit, or submission, to mean the same thing as a job.

Jobs are programs that are executed at an appropriate point on the grid. They may compute something, execute one or more system commands, move or collect data, or operate machinery. A grid application that is organized as a collection of jobs is usually designed to have these jobs execute in parallel on different machines in the grid.

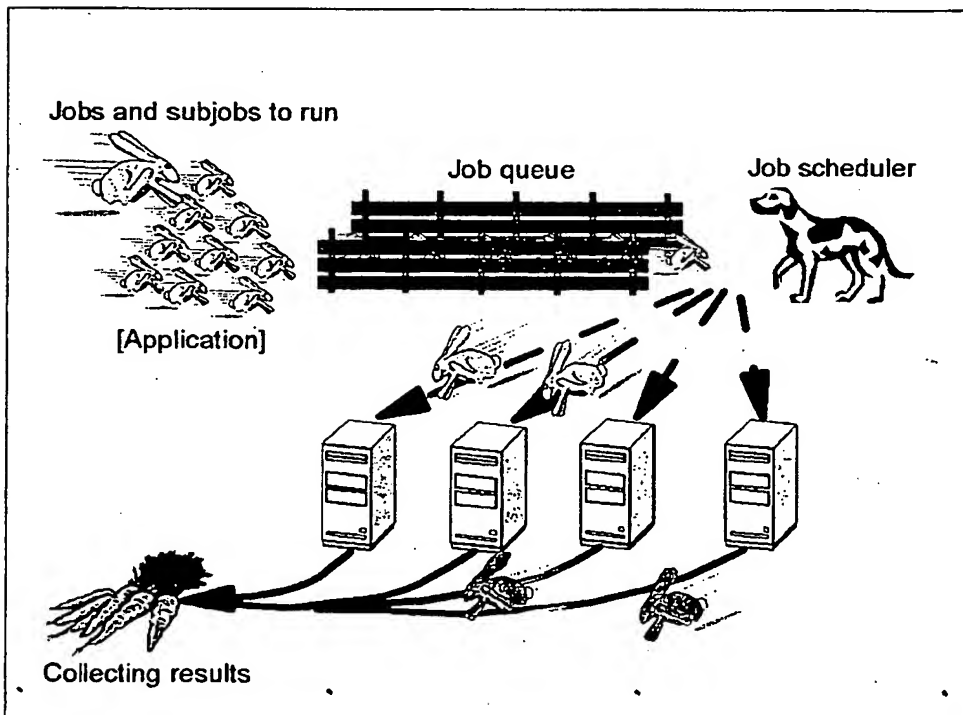


Figure 6 An application is one or more jobs that are scheduled to run on machines in the grid; the results are collected and assembled to produce the answer

The jobs may have specific dependencies that may prevent them from executing in parallel in all cases. For example, they may require some specific input data that must be copied to the machine on which the job is to run. Some jobs may require the output produced by certain

other jobs and cannot be executed until those prerequisite jobs have completed executing. Jobs may spawn additional subjobs, depending on the data they process. This work flow can create a hierarchy of jobs and subjobs. Finally, the results of all of the jobs must be collected and appropriately assembled to produce the ultimate answer for the application.

Scheduling, reservation, and scavenging

The grid system is responsible for sending a job to a given machine to be executed. In the simplest of grid systems, the user may select a machine suitable for running his job and then execute a grid command that sends the job to the selected machine. More advanced grid systems would include a job "scheduler" of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Schedulers react to current availability of resources on the grid. The term "scheduling" is not to be confused with "reservation" of resources in advance to improve the quality of service. Sometimes the term "resource broker" is used in place of "scheduler," but this term implies that some sort of bartering capability is factored into scheduling.

In a "scavenging" grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job that is satisfied by the machine's resources. Scavenging is usually implemented in a way that is unobtrusive to the normal machine user. If the machine becomes busy with local non-grid work, the grid job is usually suspended or delayed. This situation creates somewhat unpredictable completion times for grid jobs, although it is not disruptive to those machines donating resources to the grid.

To create more predictable behavior, grid machines are often "dedicated" to the grid and are not preempted by outside work. This enables schedulers to compute the approximate completion time for a set of jobs, when their running characteristics are known.

As a further step, grid resources can be "reserved" in advance for a designated set of jobs. Such reservations operate much like a calendaring system used to reserve conference rooms for meetings. This is done to meet deadlines and guarantee quality of service. When policies permit, resources reserved in advance could also be scavenged to run lower priority jobs when they are not busy during a reservation period, yielding to jobs for which they are reserved. Thus, various combinations of scheduling, reservation, and scavenging can be used to more completely utilize the grid.

Scheduling and reservation is fairly straightforward when only one resource type, usually CPU, is involved. However, additional grid optimizations can be achieved by considering more resources in the scheduling and reservation process. For example, it would be desirable to assign executing jobs to machines nearest to the data that these jobs require. This would reduce network traffic and possibly reduce scalability limits. Optimal scheduling, considering multiple resources, is a difficult mathematics problem. Therefore, such schedulers may use heuristics. These heuristics are rules that are designed to improve the probability of finding the best combination of job schedules and reservations to optimize throughput or any other metric.

Intragrid to Intergrid

There have been attempts to formulate a precise definition for what a "grid" is. In fact, the concept of grid computing is still evolving and most attempts to define it precisely end up excluding implementations that many would consider to be grids. We will be pragmatic and not claim to make any definitive descriptions of what a grid is and is not. Therefore, the following descriptions of various kinds of "grids" must be taken loosely.

Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as a hierarchy spanning the world. In this section, we will describe some examples in this range of grid system topologies.

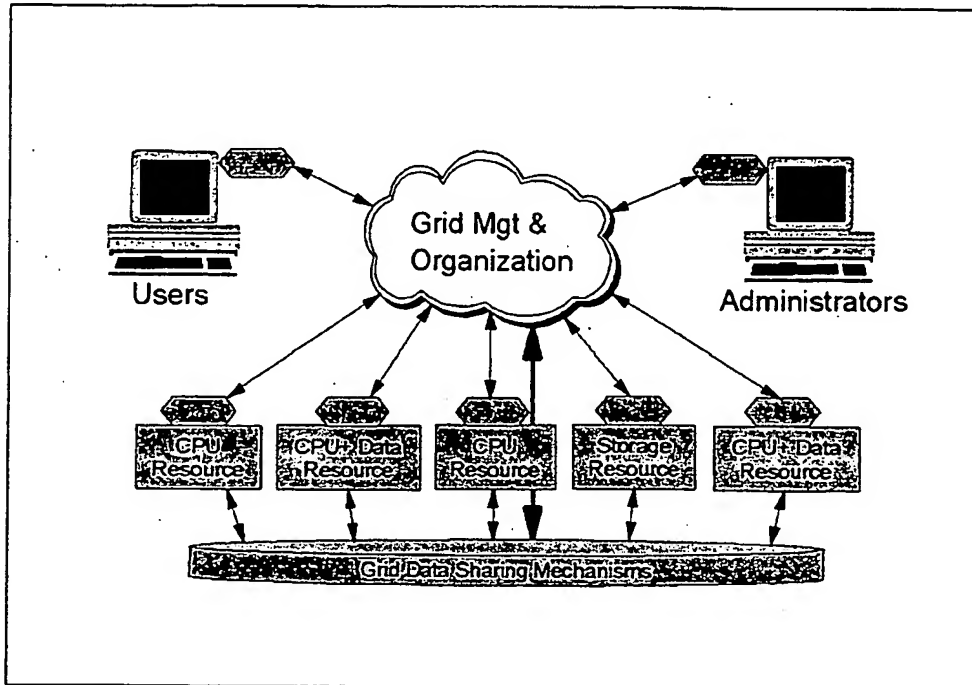


Figure 7 A simple grid

As presented in Figure 7, the simplest grid consists of just a few machines, all of the same hardware architecture and same operating system, connected on a local network. This kind of grid uses homogeneous systems so there are fewer considerations and may be used just for experimenting with grid software. The machines are usually in one department of an organization, and their use as a grid may not require any special policies or security concerns. Because the machines have the same architecture and operating system, choosing application software for these machines is usually simple. Some people would call this a "cluster" implementation rather than a "grid."

The next progression would be to include heterogeneous machines. In this configuration, more types of resources are available. The grid system is likely to include some scheduling components. File sharing may still be accomplished using networked file systems. Machines participating in the grid may include ones from multiple departments but within the same organization. Such a grid is also referred to as an "Intragrid."

As the grid expands to many departments, policies may be required for how the grid should be used. For example, there may be policies for what kinds of work is allowed on the grid and at what times. There may be a prioritization by department or by kinds of applications that should have access to grid resources. Also, security becomes more important as more organizations are involved. Sensitive data in one department may need to be protected from access by jobs running for other departments. Dedicated grid machines may be added to increase the quality of service for grid computing, rather than depending entirely on scavenged resources.

The grid may grow geographically in an organization that has facilities in different cities. Dedicated communications' connections may be used among these facilities and the grid. In some cases, VPN tunneling or other technologies may be used over the Internet to connect the different parts of the organization. Security increases in importance once the bounds of any given facility are traversed. The grid may grow to be hierarchically organized to reduce the contention implied by central control, increasing scalability.

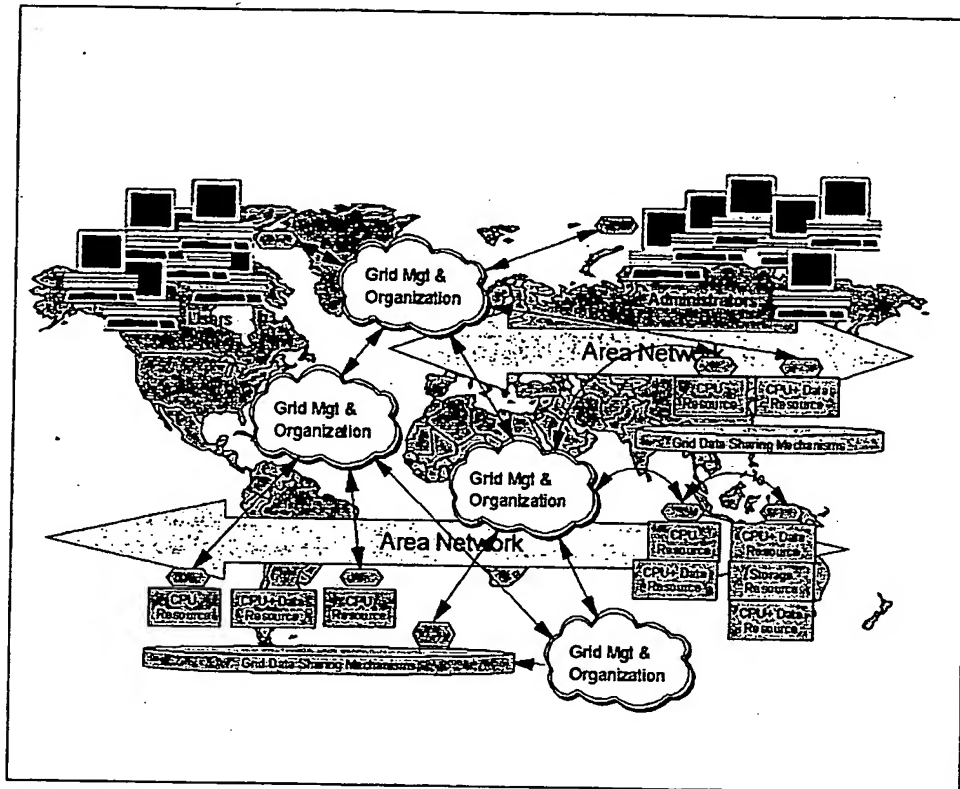


Figure 8 A more complex Intergrid

Over time, as illustrated in Figure 8, a grid may grow to cross organization boundaries, and may be used to collaborate on projects of common interest. This is known as an "Intergrid." The highest levels of security are usually required in this configuration to prevent possible attacks and spying. The Intragrid offers the prospect for trading or brokering resources over a much wider audience. Resources may be purchased as a utility from trusted suppliers.

Grid construction

An *ad hoc* grid may be installed by a few programmers in their spare time, but as the grid grows, and as users become more dependent on it for mission-critical work, a degree of planning is essential. It is best to understand the organization's requirements and choose grid technologies that best fit these requirements. This section discussed some of the planning considerations and grid components that address the requirements.

Deployment planning

The use of a grid is often born from a need for increased resources of some type. One often looks to their neighbor who may have excess capacity in the particular resource. One of the first considerations is the hardware available and how it is connected via a LAN or WAN. Next, an organization may want to add additional hardware to augment the capabilities of the grid. It is important to understand the applications to be used on the grid. Their characteristics can affect the decisions of how to best choose and configure the hardware and its data connectivity.

Security

Security is a much more important factor in planning and maintaining a grid than in conventional distributed computing, where data sharing comprises the bulk of the activity. In a grid, the member machines are configured to execute programs rather than just move data. This makes an unsecured grid potentially fertile ground for viruses and Trojan horse programs. For this reason, it is important to understand exactly which components of the grid must be rigorously secured to deter any kind of attack. Furthermore, it is important to understand the issues involved in authenticating users and properly executing the responsibilities of a certificate authority.

Organization

The technology considerations are important in deploying a grid. However, organizational and business issues can be equally important. It is important to understand how the departments in an organization interact, operate, and contribute to the whole. Often, there are barriers built between departments and projects to protect their resources in an effort to increase the probability of timely success. However, by rethinking some of these relationships, one can find that more sharing of resources can sometimes benefit the entire organization better. For example, a project that finds itself behind schedule and over budget may not be able to afford the resources required to solve the problem. A grid would give such projects an added measure of safety, providing an extra margin of resource capacity needed to finish the project. Similarly, a project in its early stages, when computing resources are not being fully utilized, may be able to donate them to other projects in need. A grid also offers the ability for the organization's management to see the bigger priority picture and react more quickly in shifting resource utilization, priorities, and policies.

Grid software components

This section presents some of the key components that must be discussed before designing a grid computing architecture.

Management components

Any grid system has some management components. First, there is a component that keeps track of the resources available to the grid and which users are members of the grid. This information is used primarily to decide where grid jobs should be assigned.

Second, there are measurement components that determine both the capacities of the nodes on the grid and their current utilization rate at any given time. This information is used to schedule jobs in the grid. Such information is also used to determine the health of the grid, alerting personnel to problems such as outages, congestion, or overcommitment. This information is also used to determine overall usage patterns and statistics, as well as to log and account for usage of grid resources.

Third, advanced grid management software can automatically manage many aspects of the grid. This is known as "autonomic computing," or "recovery oriented computing." This

software would automatically recover from various kinds of grid failures and outages, finding alternative ways to get the workload processed.

Donor software

Each machine contributing resources typically needs to enroll as a member of the grid and install some software that manages the grid's use of its resources. Usually, some sort of identification and authentication procedure must be performed before a machine can join the grid. A certificate authority can be used to establish the identity of the donor machine as well as the users and the grid itself.

Some grid systems provide their own login to the grid while others depend on the native operating systems for user authentication. In the latter case, a user ID mapping system may be needed to match the user's rights properly on different machines. This typically is manually maintained by a grid administrator. He determines which user ID a given user may possess on each grid machine and enters these IDs in a protected data base or registry. In this way, when grid jobs are submitted to different machines for a user, the proper local machine user ID is used for determining the users rights.

In some grid systems, it is possible to join the grid without any special authentication. And in others, it is possible for any user to submit jobs to the grid. Such systems may be convenient to set up, but should be discouraged in larger deployments due to the serious security problems that they would open up.

The grid system makes information about the newly added resources available throughout the grid. The donor machine will usually have some sort of monitor that determines or measures how busy the machine is and the rate or amount of resources utilized. This information is "bubbled up" to the management software of the grid and used to schedule use of those resources accordingly. In a scavenging system, this information tells the grid management software when the machine is idle and available for work.

Most importantly, the software installed on a given machine can accept an executable job from the grid management system and execute it. A user somewhere on the grid submits a job for execution on the grid. The grid management software must communicate with the grid donor software to send the job there. The donor grid software must be able to receive the executable file or select the proper one from copies pre-installed on the donor machine. The software is executed and the output is sent back to the requester. More advanced implementations can dynamically adjust the priority of a running job, suspend it and resume it later, or checkpoint it with the possibility of resuming its execution on a different machine. These kinds of actions may be necessary to respond to load balancing problems or priority or policy changes in the grid.

Submission software

Usually any member machine of a grid can be used to submit jobs to the grid and initiate grid queries. However, in some grid systems, this function is implemented as a separate component installed on "submission nodes" or "submission clients." When a grid is built using dedicated resources rather than scavenged resources, separate submission software is usually installed on the user's desktop or workstation.

Distributed grid management

Larger grids may have a hierarchical or other type of organizational topology usually matching the connectivity topology. That is, machines locally connected together with a LAN form a "cluster" of machines. The grid may be organized in a hierarchy consisting of clusters of clusters. The work involved in managing the grid is distributed to increase the scalability of the grid. The collection and grid operation and resource data as well as job scheduling is distributed to match the topology of the grid. For example, a central job scheduler will not

schedule a submitted job directly to the machine which is to execute it. Instead the job is sent to a lower level scheduler which handles a set of machines (or further clusters). The lower level scheduler handles the assignment to the specific machine. Similarly, the collection of statistical information is distributed. Lower level clusters receive activity information from the individual machines, aggregate it, and send it to higher level management nodes in the hierarchy.

Schedulers

Most grid systems include some sort of job scheduling software. This software locates a machine on which to run a grid job that has been submitted by a user. In the simplest cases, it may just blindly assign jobs in a round-robin fashion to the next machine matching the resource requirements. However, there are advantages to using a more advanced scheduler.

Some schedulers implement a job priority system. This is sometimes done by using several job queues, each with a different priority. As grid machines become available to execute jobs, the jobs are taken from the highest priority queues first. Policies of various kinds are also implemented using schedulers. Policies can include various kinds of constraints on jobs, users, and resources. For example, there may be a policy that restricts grid jobs from executing at certain times of the day.

Schedulers usually react to the immediate grid load. They use measurement information about the current utilization of machines to determine which ones are not busy before submitting a job. Schedulers can be organized in a hierarchy. For example, a meta-scheduler may submit a job to a cluster scheduler or other lower level scheduler rather than to an individual machine.

More advanced schedulers will monitor the progress of scheduled jobs managing the overall work-flow. If the jobs are lost due to system or network outages, a good scheduler will automatically resubmit the job elsewhere. However, if a job appears to be in an infinite loop and reaches a maximum timeout, then such jobs should not be rescheduled. Typically, jobs have different kinds of completion codes, some of which are suitable for re-submission and some of which are not.

Reserving resources on the grid in advance is accomplished with a "reservation system." It is more than a scheduler. It is first a calendar based system for reserving resources for specific time periods and preventing any others from reserving the same resource at the same time. It also must be able to remove or suspend jobs that may be running on any machine or resource when the reservation period is reached.

Communications

A grid system may include software to help jobs communicate with each other. For example, an application may split itself into a large number of subjobs. Each of these subjobs is a separate job in the grid. However, the application may implement an algorithm that requires that the subjobs communicate some information among them. The subjobs need to be able to locate other specific subjobs, establish a communications connection with them, and send the appropriate data. The open standard Message Passing Interface (MPI) and any of several variations is often included as part of the grid system for just this kind of communication.

Observation, management, and measurement

We mentioned above the schedulers react to current loads on the grid. Usually, the donor software will include some tools that measure the current load and activity on a given machine using either operating system facilities or by direct measurement. This software is sometimes referred to as a "load sensor." Some grid systems provide the means for implementing custom load sensors for other than CPU or storage resources.

Such measurement information is useful not only for scheduling, but also for discovering overall usage patterns in the grid. The statistics can show trends which may signal the need for additional hardware. Also, measurement information about specific jobs can be collected and used to better predict the resource requirements of that job the next time it is run. The better the prediction, the more efficiently the grid's workload can be managed.

The measurement information can also be saved for accounting purposes, to form the basis for grid resource brokering, or to manage priorities more fairly. The information can also be displayed in various forms to better visualize grid activity and utilization.

Using a grid: A user's perspective

This section describes the typical usage activities in using the grid from an user's perspective.

Enrolling and installing grid software

A user first enrolls as a grid user, and installs the provided grid software on his own machine. He may optionally enroll his machine as a donor on the grid.

Enrolling in the grid may require authentication for security purposes. The user positively establishes his identity with a certificate authority. This should not be done solely via the Internet. The certificate authority must take steps to assure that the user is in fact who he claims to be. The certificate authority makes a special certificate available to software needing to check the true identity of a grid user and his grid requests. Similar steps may be required to identify the donating machine. The user has the responsibility of keeping his grid credentials secure.

Once the user and/or machine are authenticated, the grid software is provided to the user for installing on his machine for the purposes of using the grid as well as donating to the grid. This software may be automatically preconfigured by the grid management system to know the communication address of the management nodes in the grid and user or machine identification information. In this way, the installation may be a one click operation with a minimum of interaction required on the part of the user. In less automated grid installations, the user may be asked to identify the grid's management node and possibly other configuration information. He may choose to limit the resources donated to the grid, the times that his machine is usable by the grid, and other policy related constraints. The user may also need to inform the grid administrator which user IDs are his on other machines that exist on the grid.

Logging onto the grid

To use the grid, most grid systems require the user to log on to a system using a user ID that is enrolled in the grid. Other grid systems may have their own grid login ID separate from the one on the operating system. A grid login is usually more convenient for grid users. It eliminates the ID matching problems among different machines. To the user, it makes the grid look more like one large virtual computer rather than a collection of individual machines. Globus, for example, implements a proxy login model that keeps the user logged in for a specified amount of time, even if he logs off and back on the operating system and even if the machine is rebooted.

Once logged on, the user can query the grid and submit jobs. Some grid implementations permit some query functions if the user is not logged into the grid or even if the user is not enrolled in the grid.

Queries and submitting jobs

The user will usually perform some queries to check to see how busy the grid is, to see how his submitted jobs are progressing, and to look for resources on the grid. Grid systems usually provide command line tools as well as graphical user interfaces (GUIs) for queries. Command line tools are especially useful when the user wants to write a script that automates a sequence of actions. For example, the user might write a script to look for an available resource, submit a job to it, watch the progress of the job, and present the results when the job has finished.

Job submission usually consists of three parts, even if there is only one command required. First, some input data and possibly the executable program or execution script file are sent to the machine to execute the job. Sending the input is called "staging the input data." Alternatively, the data and program files may be pre-installed on the grid machines or accessible via a mountable networked file system. When the grid consists of heterogeneous machines, there may be multiple executable program files, each compiled for the different machine platforms on the grid. A nice feature provided by some grid systems is to register these multiple versions of the program so that the grid system can automatically choose a correctly matching version to the grid machine that will run the program. Some grid technologies require that the program and input data be first processed or "wrapped" in some way by the grid system. This may be done to add protective execution controls around the application or just to simply collect all of the data files into one.

Second, the job is executed on the grid machine. The grid software running on the donating machine executes the program in a process on the user's behalf. It may use a common user ID on the machine or it may use the user's own user ID, depending on which grid technology is used. Some grid systems implement a protective "sandbox" around the program so that it cannot cause any disruption to the donating machine if it encounters a problem during execution. Rights to access files and other resources on the grid machine may be restricted.

Third, the results of the job are sent back to the submitter. In some implementations, intermediate results can be viewed by the user who submitted the job. In some grid technologies that do not automatically stage the output data back to the user, the results must be explicitly sent to the user, perhaps using a networked file system.

Scripts are also useful for submitting a series of jobs, for a parameter space application, for example. Some computation problems consist of a search for the desired result based on some input parameters. The goal is to find the input parameters that produce the best desired result. For each input parameter, a separate job is executed to find the result for that value. The whole application consists of many such jobs, which explore the results for a large number of input parameter values. Scripts are usually used to launch the many subjobs, each receiving their own particular parameter values. Parameter inputs can sometimes be more complex than simply a number. Sometimes a different input data set represents the "input parameter." Scripts help automate the large variety of more complex parameter space study problems. For simpler parameter space inputs, some grid products provide a GUI to submit the series of subjobs, each with different input parameter values.

When there are a large number of subjobs, the work required to collect the results and produce the final result is usually accomplished by a single program, usually running on the machine at the point of job submission. If there are a very large number subjobs required for an application, the work of collecting the results might be distributed as well. For example, the subjob that submits more subjobs to the grid would be responsible for collecting and aggregating the results of the subjobs it spawned.

Data configuration

The data accessed by the grid jobs may simply be staged in and out by the grid system. However, depending on its size and the number of jobs, this can potentially add up to a large amount of data traffic. For this reason, some thought is usually given on how to arrange to have the minimum of such data movement on the grid.

For example, if there will be a very large number of sub-jobs running on most of the grid systems for an application that will be repeatedly run, the data they use may be copied to each machine and reside until the next time the application runs. This is preferable to using a networked file system to share this data, because in such a file system, the data would be effectively moved from a central location every time the application is run. Thus is true unless the file system implements a caching feature or replicates the data automatically.

There are many considerations in efficiently planning the distribution and sharing of data on a grid. This type of analysis is necessary for large jobs to better utilize the grid and not create unnecessary bottlenecks.

Monitoring progress and recovery

The user can query the grid system to see how his application and its subjobs are progressing. When the number of subjobs becomes large, it becomes too difficult to list them all in a graphical window. Instead, there may simply be a one large bar graph showing some averaged progress metric. It becomes more difficult for the user to tell if any particular subjob is not running properly.

A grid system, in conjunction with its job scheduler, often provides some degree of recovery for subjobs that fail. A job may fail due to a:

- ▶ Programming error: The job stops part way with some program fault.
- ▶ Hardware or power failure: The machine or devices being used stop working in some way.
- ▶ Communications interruption: A communication path to the machine has failed or is overloaded with other data traffic.
- ▶ Excessive slowness: The job might be in an infinite loop or normal job progress may be limited by another process running at a higher priority or some other form of contention.

It is not always possible to automatically determine if the reason for a job's failure is due to a problem with the design of the application or if it is due to failures of various kinds in the grid system infrastructure. Schedulers are often designed to categorize job failures in some way and automatically resubmit jobs so that they are likely to succeed, running elsewhere on the grid. In some systems, the user is informed about any job failures and the user must decide whether to issue a command to attempt to rerun the failed jobs.

Grid applications can be designed to automate the monitoring and recovery of their own subjobs using functions provided by the grid system software application programming interfaces (APIs).

Reserving resources

To improve the quality of a service, the user may arrange to reserve a set of resources in advance for his exclusive or high priority use. A calendaring system analogy can be used here. Such a reservation system can also be used in conjunction with planned hardware or software maintenance events, when the affected resource might not be available for grid use.

In a scavenging grid, it may not be possible to reserve specific machines in advance. Instead, the grid management systems may allocate a larger fraction of its capacity for a given reservation to allow for the likelihood of some of the resources becoming unavailable. This must be done in conjunction with tools that have profiled the grid's workload capacity sufficiently to have reliable statistics about the grid's ability to serve the reservation.

Using a grid: An administrator's perspective

This section describes the typical usage activities in using the grid from an administrator's perspective.

Planning

The administrator should understand the organization's requirements for the grid to better choose the grid technologies that satisfy those requirements. The following sections briefly describe the steps the administrator may take to manage the grid. It is suggested that one should start by deploying a small grid first, to learn about its installation and management, before having to confront more complicated issues involved with a large grid.

Installation

First, the selected grid system must be installed on an appropriately configured set of machines. These machines should be connected using networks with sufficient bandwidth to other machines on the grid. Of prime importance is understanding the fail-over scenarios for the given grid system so that the grid can continue operating even if any of the management machines fails in some way. Machines should be configured and connected to facilitate recovery scenarios. Any critical data bases or other data essential for keeping track of the jobs in the grid, members of the grid, and machines on the grid should have suitable backups. Furthermore, public key certificates must be backed up and the private keys must be held in a highly secured place inaccessible by anyone else.

After installation, the grid software may need to be configured for the local network address and IDs. The administrator will usually require root access to the machines managing the grid. In some grid systems, he will also need root access to the donor machines be required to install the software on those as well. The software to be installed on the donor machines may need to be customized so that it can find the grid management machines automatically and include pre-installed public keys for the grid. This software may be provided to potential donors on an FTP or equivalent server or be made available on physical media.

Once, the grid is operational, there may be application software and data that should be installed on donor machines as well. This software may have specific licensing restrictions that should be understood and adhered to. Some grid systems include tools to assist with grid-wide license management. This can both help in following the rules of the licenses and most efficiently exploit those licenses.

Managing enrollment of donors and users

An ongoing task for the grid administrator is to manage the members of the grid, both the machines donating resources and the users. Users may be further organized as project groups. The administrator is responsible for controlling the rights of the users in the grid. Donor machines may have access rights that require management as well. Grid jobs running on donor machines may be executed under a special grid user ID on behalf of the users

submitting the jobs. The rights of these grid user IDs must be properly set so that grid jobs do not allow access to parts of the donor machine to which the users are not entitled.

As users join the grid, their identity must be positively established and entered in the certificate authority. The user and his certificate credentials must be added to the user list using the software appropriate for the grid system deployed. In some cases, the administrator must propagate the user information to several or all grid machines. Also, when the grid system depends primarily on the operating system for user login, the administrator may need to add entries to map the grid user to specific operating system user IDs on the donor machines.

Similar enrollment activity is usually required to enroll donor machines into the grid. The machine's identity is established and registered with the certificate authority. The administrator of the grid must have an agreement with the administrator of the donor machine about user IDs, software, access rights, and any policy restrictions. The administrator must enter the machine's identification credentials, addresses, and resource characteristics using the appropriate software for enrolling the donor machine into the grid. In some cases, the administrator may need to manually propagate this information to other machines in the grid.

Corresponding procedures for removing users and machines must be executed by the administrator.

Certificate authority

It is critical to ensure the highest levels of security in a grid because the grid is designed to execute code and not just share data. Thus, it can be fertile ground for viruses, Trojan horses, and other attacks if the grid system is compromised in any way. The certificate authority is one of the most important aspects of maintaining strong grid security. An organization may choose to use an external certificate authority or operate one itself. You must be able to trust the certificate authority to strictly adhere to its responsibilities.

The primary responsibilities of a certificate authority are:

- ▶ Positively identify entities requesting certificates
- ▶ Issuing, removing, and archiving certificates
- ▶ Protecting the certificate authority server
- ▶ Maintaining a namespace of unique names for certificate owners
- ▶ Serve signed certificates to those needing to authenticate entities
- ▶ Logging activity

Briefly, a certificate authority is based on the public key encryption system. In this system, keys are generated in pairs, a public key and a private key. Either one can be used to encrypt some data such that the other is needed to decrypt it. The private key is guarded by the owner and never revealed to anyone. The public one is given to anyone needing it. A certificate authority is used to hold these public keys and to guarantee who they belong to. When a user uses his private key to encrypt something, the receiver uses the corresponding public key to decrypt it. The receiver knows that only that user's public key can decrypt the message correctly. However, anyone could intercept this message and decrypt it because anyone can get the originator's public key. If the originator instead doubly encrypts the message with his private key and the intended recipient's public key, a secure communication link is formed. The receiver uses his private key to decrypt the message and then uses the sender's public key for the second decryption. Now the recipient knows that if the message decrypts properly, then only the sender could have sent it and furthermore, the sender knows that only the intended receiver can decrypt it. The beauty of all of this is that nobody had to

securely carry an encryption key from the sender to the receiver, as must be done for conventional encryption systems, and any tampering with the communication is revealed. A similar exchange is used to get anyone's public key from the certificate authority, so that the user knows that he has received an unaltered public key for the desired user.

Resource management

Another responsibility of the administrator is to manage the resources of the grid. This includes setting permissions for grid users to use the resources as well as tracking resource usage and implementing a corresponding accounting or billing system. Usage statistics are useful in identifying trends in an organization that may require the acquisition of additional hardware, reduction in excess hardware to reduce costs, and adjustments in priorities and policies to achieve utilization that is fairer or better achieves the overall goals of an organization.

Some grid components, usually job schedulers, have provisions for enforcing priorities and policies of various kinds. It is the responsibility of the administrator to configure these to best meet the goals of the overall organization. Software license managers can be used in a grid setting to control the proper utilization. These may be configured to work with job schedulers to prioritize the use of the limited licenses.

Data sharing

For small grids, the sharing of data can be fairly easy, using existing networked file systems, databases, or standard data transfer protocols. As a grid grows and the users become dependent on any of the data storage repositories, the administrator should consider procedures to maintain backup copies and replicas to improve performance. All of the resource management concerns apply to data on the grid.

Using a grid: An application developer's perspective

Grid applications can be categorized in one of the following three categories:

- ▶ Applications that are not enabled for using multiple processors but can be executed on different machines.
- ▶ Applications that are already designed to use the multiple processors of a grid setting.
- ▶ Applications that need to be modified or rewritten to better exploit a grid.

The latter category is of interest to grid application developers. They will find a need for tools for debugging and measuring the behavior of grid applications. Such grid based tools are still in their infancy. It may be useful for developers to configure a small grid of their own so that they can use debuggers on each machine to control and watch the detailed workings of the applications. Since the debugging process can bypass certain security precautions, it may not always be wise to allow such debugging on a production grid.

Globus is more a developer's toolkit for building grid components rather than a comprehensive grid system. It has the basic components needed to build new facilities to manage grid operations, measurement, repair, and debug grid applications. Tools conforming to the emerging Open Grid Services Architecture (OGSA) interfaces will be usable on various vendor grid systems.

The present and the future

The Globus toolkit is a set of tools useful for building a grid. Its strength is a good security model, with a provision for hierarchically collecting data about the grid, as well as the basic facilities for implementing a simple, yet world-spanning grid. Globus will grow over time through the work of many organizations that are extending its capabilities. More information about Globus can be obtained at <http://www.globus.org>.

Most grid systems include some job schedulers, but as grids span wider areas, there will be a need for more meta-schedulers that can manage variously configured collections of clusters and smaller grids. These schedulers will evolve to better schedule jobs, considering multiple resources rather than just CPU utilization. They will also extend their reach to implement better quality of service, using reservations, redundancy, and history profiles of jobs and grid performance.

Today, grid systems are still at the early stages of providing a reliable, well performing, and automatically recoverable virtual data sharing and storage. We will see products that take on this task in a grid setting, federating data of all kinds, and achieving better performance, integration with scheduling, reliability, and capacity.

Autonomic computing has the goal to make the administrator's job easier by automating the various complicated tasks involved in managing a grid. These include identifying problems in real time and quickly initiating corrective actions before they seriously impair the grid.

Open Grid Services Architecture (OGSA) is an open standard at the base of all of these future grid enhancements. OGSA will standardize the grid interfaces that will be used by the new schedulers, autonomic computing agents, and any number of other services yet to be developed for the grid. It will make it easier to assemble the best products from various vendors, increasing the overall value of grid computing. More information about OGSA can be obtained at <http://www.globus.org/ogsa>.

What the grid cannot do

A word of caution should be given to the overly enthusiastic. The grid is not a silver bullet that can take any application and run it a 1000 times faster without the need for buying any more machines or software. Not every application is suitable or enabled for running on a grid. Some kinds of applications simply cannot be parallelized. For others, it can take a large amount of work to modify them to achieve faster throughput. The configuration of a grid can greatly affect the performance, reliability, and security of an organization's computing infrastructure. For all of these reasons, it is important for the us to understand how far the grid has evolved today and which features are coming tomorrow or in the distant future.

Acknowledgements

This publication was produced during the grid computing project, organized and coordinated by the International Technical Support Organization in October 2002. The following team of specialists, from around the world, contributed to this project:

Jonathan Armstrong and Viktors Berstis
Grid Computing Initiative, e-Technology Center - IBM Austin

Mike Kendzierski
ITS Northeast Region - IBM North America

Andreas Neukoetter
iBM @server OGSA Development - IBM Germany

Richard Bing-Wo
Professional Services Consultant - IBM North America

Olegario Hernandez
Business Partner - IBM Chile

Masanobu Takagi
IBM Japan

Adeeb Amir
Professional Services Consultant - IBM North America

Ryo Murakawa
IBM Japan

Norbert Bieberstein
Solution Development Manager - IBM Germany

Luis Ferreira
Grid and Linux Team, International Technical Support Organization - IBM Austin

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

This document created or updated on November 11, 2002.




Send us your comments in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:
ibm.com/redbooks
- ▶ Send your comments in an Internet note to:
redbook@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AFS®
IBM®
IBM eServer™

DFS™
Redbooks™
Redbooks(logo)™ 

PowerPC®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Globus Project™ and Globus Toolkit™ are trademarks held by the University of Chicago.

Linux is a registered trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.

A Resource Management Architecture for Metacomputing Systems

Karl Czajkowski * Ian Foster † Nick Karonis † Carl Kesselman * Stuart Martin †
Warren Smith † Steven Tuecke †

{karlcz, itf, karonis, carl, smartin, wsmith, tuecke}@globus.org
<http://www.globus.org>

*Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292-6695
<http://www.isi.edu>

†Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
<http://www.mcs.anl.gov>

Abstract

Metacomputing systems are intended to support remote and/or concurrent use of geographically distributed computational resources. Resource management in such systems is complicated by five concerns that do not typically arise in other situations: site autonomy and heterogeneous substrates at the resources, and application requirements for policy extensibility, co-allocation, and online control. We describe a resource management architecture that addresses these concerns. This architecture distributes the resource management problem among distinct local manager, resource broker, and resource co-allocator components and defines an extensible resource specification language to exchange information about requirements. We describe how these techniques have been implemented in the context of the Globus metacomputing toolkit and used to implement a variety of different resource management strategies. We report on our experiences applying our techniques in a large testbed, GUSTO, incorporating 15 sites, 390 computers, and 3600 processors.

1 Introduction

Metacomputing systems allow applications to assemble and use collections of computational resources on an as-needed basis, without regard to physical location. Various groups are implementing such systems and exploring applications in distributed supercomputing, high-throughput computing, smart instruments, collaborative environments, and data mining [10, 12, 18, 20, 22, 6, 25].

This paper is concerned with *resource management* for

metacomputing: that is, with the problems of locating and allocating computational resources, and with authentication, process creation, and other activities required to prepare a resource for use. We do not address other issues that are traditionally associated with scheduling (such as decomposition, assignment, and execution ordering of tasks) or the management of other resources such as memory, disk, and networks.

The metacomputing environment introduces five challenging resource management problems: site autonomy, heterogeneous substrate, policy extensibility, co-allocation, and online control.

1. The *site autonomy* problem refers to the fact that resources are typically owned and operated by different organizations, in different administrative domains [5]. Hence, we cannot expect to see commonality in acceptable use policy, scheduling policies, security mechanisms, and the like.
2. The *heterogeneous substrate* problem derives from the site autonomy problem and refers to the fact that different sites may use different local resource management systems [16], such as Condor [18], NQE [1], CODINE [11], EASY [17], LSF [28], PBS [14], and LoadLeveler [15]. Even when the same system is used at two sites, different configurations and local modifications often lead to significant differences in functionality.
3. The *policy extensibility* problem arises because metacomputing applications are drawn from a wide range of domains, each with its own requirements. A resource management solution must support the frequent development of new domain-specific manage-

ment structures, without requiring changes to code installed at participating sites.

4. The *co-allocation* problem arises because many applications have resource requirements that can be satisfied only by using resources simultaneously at several sites. Site autonomy and the possibility of failure during allocation introduce a need for specialized mechanisms for allocating multiple resources, initiating computation on those resources, and monitoring and managing those computations.
5. The *online control* problem arises because substantial negotiation can be required to adapt application requirements to resource availability, particularly when requirements and resource characteristics change during execution. For example, a tele-immersive application that needs to simulate a new entity may prefer a lower-resolution rendering, if the alternative is that the entity not be modeled at all. Resource management mechanisms must support such negotiation.

As we explain in Section 2, no existing resource management systems addresses all five problems. Some batch queuing systems support co-allocation, but not site autonomy, policy extensibility, and online control [16]. Condor supports site autonomy, but not co-allocation or online control [18]. Gallop [26] addresses online control and policy extensibility, but not the heterogeneous substrate or co-allocation problem. Legion [12] does not address the heterogeneous substrate problem.

In this paper, we describe a resource management architecture that we have developed to address the five problems. In this architecture, developed in the context of the Globus project [10], we address problems of site autonomy and heterogeneous substrate by introducing entities called *resource managers* to provide a well-defined interface to diverse local resource management tools, policies, and security mechanisms. To support online control and policy extensibility, we define an extensible *resource specification language* that supports negotiation between different components of a resource management architecture, and we introduce *resource brokers* to handle the mapping of high-level application requests into requests to individual managers. We address the problem of co-allocation by defining various co-allocation strategies; which we encapsulate in *resource co-allocators*.

One measure of success for an architecture such as this is its usability in a practical setting. To this end, we have implemented and deployed this architecture on GUSTO, a large computational grid testbed comprising 15 sites, 330 computers, and 3600 processors, using LSF, NQE, LoadLeveler, EASY, Fork, and Condor as local schedulers. To date, this architecture and testbed have been

used by ourselves and others to implement numerous applications and half a dozen different higher-level resource management strategies. This experiment represents a significant step forward in terms of number of global meta-computing services implemented and number and variety of commercial and experimental local resource management systems employed. A more quantitative evaluation of the approach remains as a significant challenge for future work.

The rest of this paper is structured as follows. In the next section, we review current distributed resource management solutions. In subsequent sections we first outline our architecture and then examine each major function in detail: the resource specification language, local resource managers, resource brokers, and resource co-allocators. We summarize the paper and discuss future work in Section 8.

2 Resource Management Approaches

Previous work on resource management for metacomputing systems can be broken into two broad classes:

- *Network batch queuing systems.* These systems focus strictly on resource management issues for a set of networked computers. These systems do not address policy extensibility and provide only limited support for online control and co-allocation.
- *Wide-area scheduling systems.* Here, resource management is performed as a component of mapping application components to resources and scheduling their execution. To date, these systems do not address issues of heterogeneous substrates, site autonomy, and co-allocation.

In the following, we use representative examples of these two types of system to illustrate the strengths and weaknesses of current approaches.

2.1 Networked Batch Queuing Systems

Networked batch queuing systems, such as NQE [1], CO-DINE [11], LSF [28], PBS [14], and LoadLeveler [15], handle user-submitted jobs by allocating resources from a networked pool of computers. The user characterizes application resource requirements either explicitly, by some type of job control language, or implicitly, by selecting the queue to which a request is submitted. Networked batch queuing systems typically are designed for single administrative domains, making site autonomy difficult to achieve. Likewise, the heterogeneous substrate problem is also an issue because these systems generally assume

that they are the only resource management system in operation. One exception is the CODINE system, which introduces the concept of a *transfer queue* to allow jobs submitted to CODINE to be allocated by some other resource management system, at a reduced level of functionality. An alternative approach to supporting substrate heterogeneity is being explored by the PSCHED [13] initiative. This project is attempting to define a uniform API through which a variety of batch scheduling systems may be controlled. The goals of PSCHED are similar in many ways to those of the Globus Resource Allocation Manager described in Section 5.

Batch scheduling systems provide a limited form of policy extensibility in that resource management policy is set by either the system or the system administrator, by the creation of scheduling policy or batch queues. However, this capability is not available to the end users, who have little control over how the batch scheduling system interprets their resource requirements.

Finally, we observe that batch queuing systems have limited support for on-line allocation, as these systems are designed to support applications in which the requirements specifications are in the form "get *X* done soon", where *X* is precisely defined but "soon" is not. In meta-computing applications, we have more complex, fluid constraints, in which we will want to make tradeoffs between time (when) and space (physical characteristics). Such constraints lead to a need for the resource management system to provide capabilities such as negotiation, inquiry interfaces, information-based control, and co-allocation, none of which are provided in these systems.

In summary, batch scheduling systems do not provide in themselves a complete solution to metacomputing resource management problems. However, clearly some of the mechanisms developed for resource location, distributed process control, remote file access, to name a few, can be applied to wide-area systems as well. Furthermore, we note that network batch queuing systems will necessarily be part of the local resource management solution. Hence, any metacomputing resource management architecture must be able to interface to these systems.

2.2 Wide-Area Scheduling Systems

We now examine how resource management is addressed within systems developed specifically to schedule metacomputing applications. To gain a good perspective on the range of possibilities, we discuss four different schedulers, designed variously to support specific classes of applications (Gallop [26]), an extensible object-oriented system (Legion [12]), general classes of parallel programs (PRM [22]), and high-throughput computation (Condor [18]).

The Gallop [26] system allocates and schedules tasks

defined by a static task graph onto a set of networked computational resources. (A similar mechanism has been used in Legion [27].) Resource allocation is implemented by a scheduling manager, which coordinates scheduling requests, and a local manager, which manages the resources at a local site, potentially interfacing to site-specific scheduling and resource allocation services. This decomposition, which we also adopt, separates local resource management operations from global resource management policy and hence facilitates solutions to the problems of site autonomy, heterogeneous substrates, and policy extensibility. However, Gallop does not appear to handle authentication to local resource management services, thereby limiting the level of site autonomy that can be achieved.

The use of a static task-graph model makes online control in Gallop difficult. Resource selection is performed by attempting to minimize the execution time of task graph as predicted by a performance model for the application and the prospective resource. However, because the minimization procedure and the cost model is fixed, there is no support for policy extensibility. Legion [12] overcomes this limitation by leveraging its object-oriented model. Two specialized objects, an application-specific Scheduler and a resource-specific Enactor negotiate with one another to make allocation decisions. The Enactor can also provide co-allocation functions.

Gallop supports co-allocation for resources maintained within an administrative domain, but depends for this purpose on the ability to reserve resources. Unfortunately, reservation is not currently supported by most local resource management systems. For this reason, our architecture does not rely on reservation to perform co-allocation, but rather uses a separate co-allocation management service to perform this function.

The Prospero Resource Manager [22] (PRM) provides resource management functions for parallel programs written by using the PVM message-passing library. PRM consists of three components: a system manager, a job manager, and a node manager. The job manager makes allocation decisions, while the system and node manager actually allocate resources. The node manager is solely responsible for implementing resource allocation functions. Thus, PRM does not address issues of site autonomy or substrate heterogeneity. A variety of job managers can be constructed, allowing for policy extensibility, although there is no provision for composing job managers so as to extend an existing management policy. As in our architecture, PRM has both an information infrastructure (Prospero [21]) and a management API, providing the infrastructure needed to perform online control. However, unlike our architecture, PRM does not support co-allocation of resources.

Condor [18] is a resource management system designed to support high-throughput computations by discovering idle resources on a network and allocating those resources to application tasks. While Condor does not interface with existing resource management systems, resources controlled by Condor are deallocated as soon as the "rightful" owner starts to use them. In this sense, Condor supports site autonomy and heterogeneous substrates. However, Condor currently does not interoperate with local resource authentication, limiting the degree of autonomy a site can assert. Condor provides an extensible resource description language, called *classified ads*, which provides limited control over resource selection to both the application and resource. However, the matching of application component to resource is performed by a system *classifier*, which defines how matches—and consequently resource management—take place, limiting the extensibility of this selection policy. Finally, Condor provides no support for co-allocation or online control.

In summary, our review of current resource management approaches revealed a range of valuable services, but no single system that provides solutions to all five metacomputing resource management problems posed in the introduction.

3 Our Resource Management Architecture

Our approach to the metacomputing resource management problem is illustrated in Figure 1. In this architecture, an extensible *resource specification language* (RSL), discussed in Section 4 below, is used to communicate requests for resources between components: from applications to resource brokers, resource co-allocators, and resource managers. At each stage in this process, information about resource requirements, coded as an RSL expression by the application, is refined by one or more resource brokers and co-allocators; information about resource availability and characteristics is obtained from an information service.

Resource brokers are responsible for taking high-level RSL specifications and transforming them into more concrete specifications through a process we call *specialization*. As illustrated in Figure 2, multiple brokers may be involved in servicing a single request, with application-specific brokers translating application requirements into more concrete resource requirements, and different resource brokers being used to locate available resources that meet those requirements.

Transformations effected by resource brokers generate a specification in which the locations of the required resources are completely specified. Such a *ground request* can be passed to a *co-allocator*, which is responsible for

coordinating the allocation and management of resources at multiple sites. As we describe in Section 7, a variety of co-allocators will be required in a metacomputing system, providing different co-allocation semantics.

Resource co-allocators break a multirequest—that is, a request involving resources at multiple sites—into its constituent elements and pass each component to the appropriate *resource manager*. As discussed in Section 5, each resource manager in the system is responsible for taking an RSL request and translating it into operations in the local, site-specific resource management system.

The *information service* is responsible for providing efficient and pervasive access to information about the current availability and capability of resources. This information is used to locate resources with particular characteristics, to identify the resource manager associated with a resource, to determine properties of that resource, and for numerous other purposes as high-level resource specifications are translated into requests to specific managers. We use the Globus system's Metacomputing Directory Service (MDS) [8] as our information service. MDS uses the data representation and application programming interface (API) defined on the Lightweight Directory Access Protocol (LDAP) to meet requirements for uniformity, extensibility, and distributed maintenance. It defines a data model suitable for distributed computing applications, able to represent computers and networks of interest, and provides tools for populating this data model. LDAP defines a hierarchical, tree-structured name space called a *directory information tree* (DIT). Fields within the namespace are identified by a unique *distinguished name* (DN). LDAP supports both distribution and replication. Hence, the local service associated with MDS is exactly an LDAP server (or a gateway to another LDAP server, if multiple sites share a server), plus the utilities used to populate this server with up-to-date information about the structure and state of the resources within that site. The global MDS service is simply the ensemble of all these servers. An advantage of using MDS as our information service is that resource management information can be used by other tools, as illustrated in Figure 3.

4 Resource Specification Language

We now discuss the resource specification language itself. The syntax of an RSL specification, summarized in Figure 4, is based on the syntax for filter specifications in the Lightweight Directory Access Protocol and MDS. An RSL specification is constructed by combining simple parameter specifications and conditions with the operators $\&$; to specify conjunction of parameter specifications, $|$; to express the disjunction of parameter specifications, $+$; or

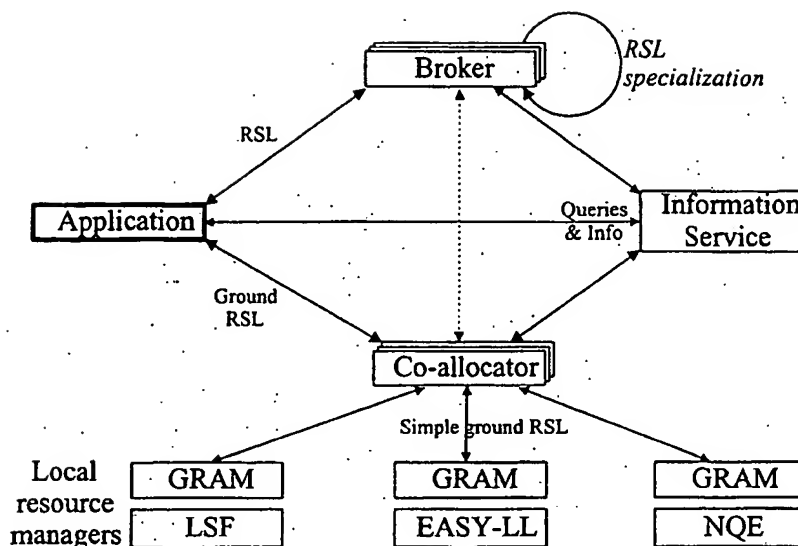


Figure 1: The Globus resource management architecture, showing how RSL specifications pass between application, resource brokers, resource co-allocators, and local managers (GRAMs). Notice the central role of the information service.

to combine two or more requests into a single compound request, or multirequest.

The set of parameter-name terminal symbols is extensible: resource brokers, co-allocators, and resource managers can each define a set of parameter names that they will recognize. For example, a resource broker that is specialized for tele-immersive applications might accept as input a specification containing a *frames-per-second* parameter and might generate as output a specification containing an *mflops-per-second* parameter, to be passed to a broker that deals with computational resources. Resource managers, the system components that actually talk to local scheduling systems, recognize two types of parameter-name terminal symbols:

- *MDS attribute names*, used to express constraints on resources: for example, *memory>=64* or *network=atm*. In this case, the parameter name refers to a field defined in the MDS entry for the resource being allocated. The truth of the parameter specification is determined by comparing the value provided with the specification with the current value associated with the corresponding field in the MDS. Arbitrary MDS fields can be specified by providing their full distinguished name.
- *Scheduler parameters*, used to communicate information regarding the job, such as *count* (number

of nodes required), *max_time* (maximum time required), *executable*, *arguments*, *directory*, and *environment* (environment variables). Schedule parameters are interpreted directly by the resource manager.

For example, the specification

```
&(executable=myprog)
  (1(&(count=5)(memory>=64))
   &(count=10)(memory>=32)))
```

requests 5 nodes with at least 64 MB memory, or 10 nodes with at least 32 MB. In this request, *executable* and *count* are scheduler attribute names, while *memory* is an MDS attribute name.

Our current RSL parser and resource manager disambiguate these two parameter types on the basis of the parameter name. That is, the resource manager knows which fields it will accept as scheduler parameters and assumes all others are MDS attribute names. Name clashes can be disambiguated by using the complete distinguished name for the MDS field in question.

The ability to include constraints on MDS attribute values in RSL specifications is important. As we discuss in Section 5, the state of resource managers is stored in MDS. Hence, resource specifications can refer to resource characteristics such as queue-length, expected wait time,

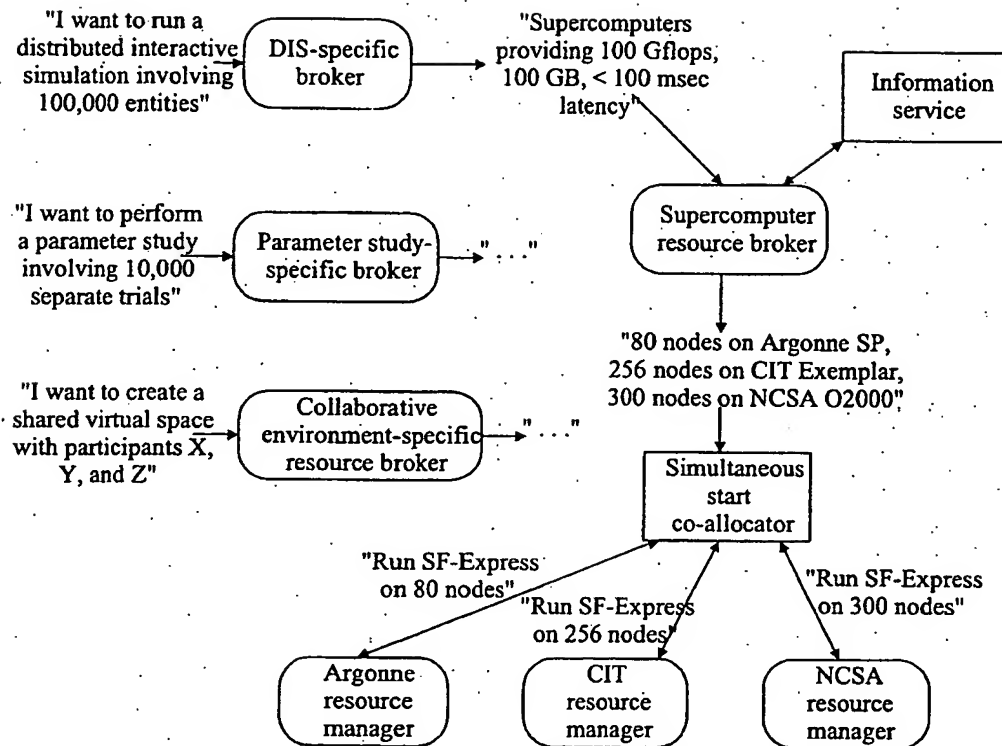


Figure 2: This view of the Globus resource management architecture shows how different types of broker can participate in a single resource request

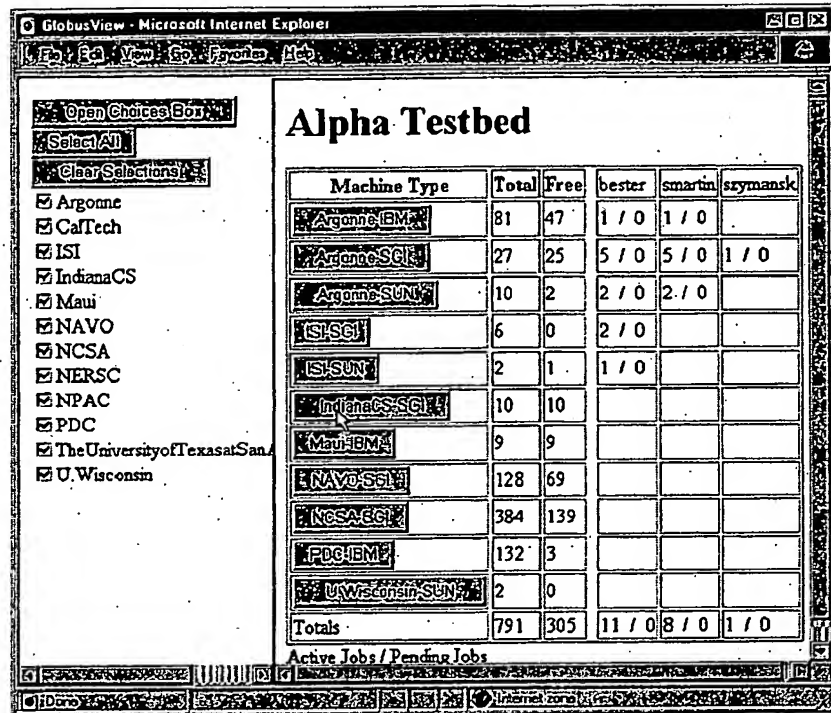


Figure 3: The GlobusView tool uses MDS information about resource manager status to present information about the current status of a metacomputing testbed. On the left, we see the sites that are currently participating in the testbed; on the right is information about the total number of nodes that each site is contributing, the number of those nodes that are currently available to external users, and the usage of those nodes by Globus users.

```

specification      := request
request            := multirequest | conjunction | disjunction | parameter
multirequest       := + request-list
conjunction        := & request-list
disjunction        := | request-list
request-list       := ( request ) request-list | ( request )
parameter         := parameter-name op value
op                 := = | > | < | >= | <= | !=
value              := ([a..Z][0..9][_])+
```

Figure 4: BNF grammar describing the syntax of an RSL request

and number of processors available. This technique provides a powerful mechanism for controlling how an RSL specification is interpreted.

The following example of a multirequest is derived from the example shown in Figure 2.

```
+(count=80)(memory>=64M)
  (executable=sf_express)
  (resourcemanager=ico16.mcs.anl.gov:8711))
(&(count=256)(network=atm)
  (executable=sf_express)
  (resourcemanager=
    neptune.cacr.caltech.edu:755))
(&(count=300)(memory>=64M)
  (executable=sf_express)
  (resourcemanager=modi4.ncsa.edu:4000))
```

This is a ground request: every component of the multirequest specifies a resource manager. A co-allocator can use the `resourcemanager` parameters specified in this request to determine to which resource manager each component of the multirequest should be submitted.

Notations intended for similar purposes include the Condor “classified ad” [18] and Chapin’s “task description vector” [5]. Our work is novel in three respects: the tight integration with a directory service, the use of specification rewriting to express broker operations (as described below), and the fact that the language and associated tools have been implemented and demonstrated effective when layered on top of numerous different low-level schedulers.

We conclude this section by noting that it is the combination of resource brokers, information service, and RSL that makes online control possible in our architecture. Together, these services make it possible to construct requests dynamically, based on current system state and negotiation between the application and the underlying resources.

5 Local Resource Management

We now describe the lowest level of our resource management architecture: the local resource managers, implemented in our architecture as Globus Resource Allocation Managers (GRAMs). A GRAM is responsible for

1. processing RSL specifications representing resource requests, by either denying the request or by creating one or more processes (a “job”) that satisfy that request;
2. enabling remote monitoring and management of jobs created in response to a resource request; and
3. periodically updating the MDS information service with information about the current availability and capabilities of the resources that it manages.

A GRAM serves as the interface between a wide area metacomputing environment and an autonomous entity able to create processes, such as a parallel computer scheduler or a Condor pool. Hence, a resource manager need not correspond to a single host or a specific computer, but rather to a service that acts on behalf of one or more computational resources. This use of local scheduler interfaces was first explored in the software environment for the I-WAY networking experiment [9], but is extended and generalized here significantly to provide a richer and more flexible interface.

A resource specification passed to a GRAM is assumed to be ground: that is, to be sufficiently concrete that the GRAM can identify local resources that meet the specification without further interaction with the entity that generated the request. A particular GRAM implementation may achieve this goal by scheduling resources itself or, more commonly, by mapping the resource specification into a request to some local resource allocation mechanisms. (To date, we have interfaced GRAMs to six different schedulers or resource allocators: Condor, EASY, Fork, LoadLeveler, LSF, and NQE.) Hence, the GRAM API plays for resource management a similar role to that played by IP for communication: it can co-exist with local mechanisms, just as IP rides on top of ethernet, FDDI, or ATM networking technology.

The GRAM API provides functions for submitting and for canceling a job request and for asking when a job (submitted or not) is expected to run. An implementation of the latter function may use queue time-estimation techniques [24]. When a job is submitted, a globally unique *job handle* is returned that can then be used to monitor and control the progress of the job. In addition, a job submission call can request that the progress of the requested job be signaled asynchronously to a supplied *callback URL*. Job handles can be passed to other processes, and callbacks do not have to be directed to the process that submitted the job request. These features of the GRAM design facilitate the implementation of diverse higher-level scheduling strategies. For example, a high-level broker or co-allocator can make a request on behalf of an application, while the application monitor the progress of the request.

5.1 GRAM Scheduling Model

We discuss briefly the scheduling model defined by GRAM because this is relevant to subsequent discussion of co-allocation. This model is illustrated in Figure 5, which shows the state transitions that may be experienced by a GRAM job.

When submitted, the job is initially *pending*, indicating that resources have not yet been allocated to the job. At some point, the job is allocated the requested

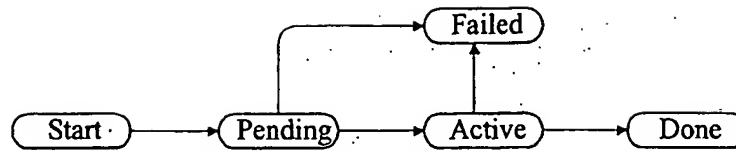


Figure 5: State transition diagram for resource allocation requests submitted to the GRAM resource management API

resources, and the application starts running. The job then transitions to the active state. At any point prior to entering the done state, the job can be terminated, causing it to enter the failed state. A job can fail because of explicit termination, an error in the format of the request, a failure in the underlying resource management system, or a denial of access to the resource. The source of the failure is provided as part of the notification of state transition. When all of the processes in the job have terminated and resources have been deallocated, the job enters the done state.

5.2 GRAM Implementation

The GRAM implementations that we have constructed have the structure shown in Figure 6. The principal components are the GRAM client library, the gatekeeper, the RSL parsing library, the job manager, and the GRAM reporter. The Globus security infrastructure (GSI) is used for authentication and for authorization.

The *GRAM client library* is used by an application or a co-allocator acting on behalf of an application. It interacts with the GRAM gatekeeper at a remote site to perform mutual authentication and transfer a request, which includes a resource specification and a callback (described below).

The *gatekeeper* is an extremely simple component that responds to a request by doing three tasks: performing mutual authentication of user and resource, determining a local user name for the remote user, and starting a job manager which executes as that local user and actually handles the request. The first two security-related tasks are performed by calls to the Globus security infrastructure (GSI), which handles issues of site autonomy and substrate heterogeneity in the security domain. To start the job manager, the gatekeeper must run as a privileged program: on Unix systems, this is achieved via `suid` or `inetd`. However, because the interface to the GSI is small and well defined, it is easy for organizations to approve (and port) the gatekeeper code. In fact, the gatekeeper code has successfully undergone security reviews at a number of large supercomputer centers. The map-

ping of remote user to locally recognized user name minimizes the amount of code that must run as a privileged program; it also allows us to delegate most authorization issues to the local system.

The *job manager* is responsible for creating the actual processes requested by the user. This task typically involves submitting a resource allocation request to the underlying resource management system, although if no such system exists on a particular resource, a simple `fork` may be performed. Once processes are created, the job manager is also responsible for monitoring the state of the created processes, notifying the callback contact of any state transitions, and implementing control operations such as process termination. The job manager terminates once the job for which it is responsible has terminated.

The *GRAM reporter* is responsible for storing into MDS various information about scheduler structure (e.g., whether the scheduler supports reservation and the number of queues) and state (e.g., total number of nodes, number of nodes currently available, currently active jobs, and expected wait time in a queue). An advantage of implementing the GRAM reporter as a distinct component is that MDS reports can continue even when no gatekeeper or job manager is running: for example, when the gatekeeper is run from `inetd`.

As noted above, GRAM implementations have been constructed for six local schedulers to date: Condor, LSF, NQE, Fork, EASY, and LoadLeveler. Much of the GRAM code is independent of the local scheduler, and so only a relatively small amount of scheduler-specific code needed to be written in each case. In most cases, this code comprises shell scripts that use the local scheduler's user-level API. State transitions are handled mostly by polling, because this proved to be more reliable than monitoring job processes by using mechanisms provided by the local schedulers.

6 Resource Brokers

As noted above, we use the term *resource broker* to denote an entity in our architecture that translates abstract re-

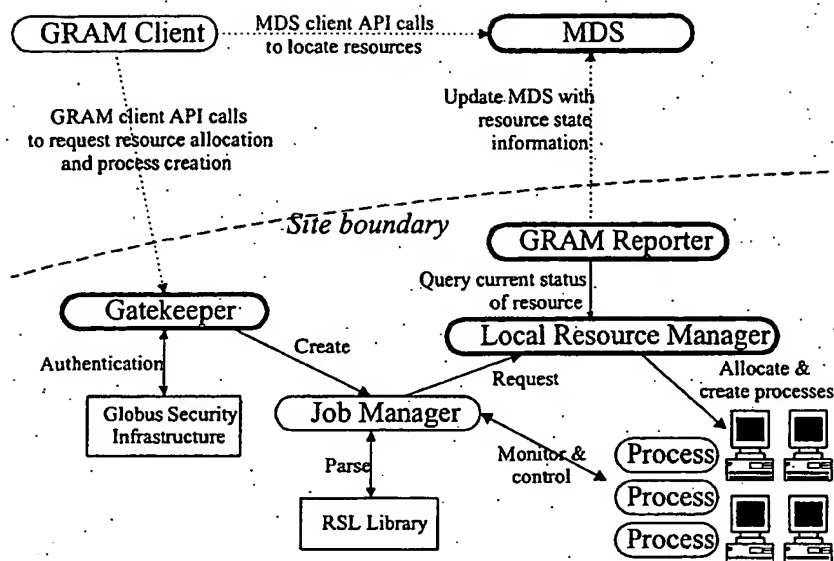


Figure 6: Major components of the GRAM implementation. Those represented by thick-lined ovals are long-lived processes, while the thin-lined ovals are short-lived processes created in response to a request.

source specifications into more concrete specifications. As illustrated in Figure 2, this definition is broad enough to encompass a variety of behaviors, including application-level schedulers [3] that encapsulate information about the types of resource required to meet a particular performance requirement, resource locators that maintain information about the availability of various types of resource, and (ultimately) traders that create markets for resources. In each case, the broker uses information maintained locally, obtained from MDS, or contained in the specification to *specialize* the specification, mapping it into a new specification that contain more detail. Requests can be passed to several brokers, effectively composing the behaviors of those brokers, until eventually the specification is specialized to the point that it identifies a specific resource manager. This specification can then be passed to the appropriate GRAM or, in the case of a multirequest, to a resource co-allocator.

We claim that our architecture makes it straightforward to develop a variety of higher-level schedulers. In support of this claim, we note that following the definition and implementation of GRAM services, a variety of people, including people not directly involved in GRAM definition, were able to construct half a dozen resource brokers quite quickly. We describe three of these here.

6.1 Nimrod-G

David Abramson and Jonathan Giddy are using GRAM mechanisms to develop Nimrod-G, a wide-area version of the Nimrod [2] tool. Nimrod automates the creation and management of large parametric experiments. It allows a user to run a single application under a wide range of input conditions and then to aggregate the results of these different runs for interpretation. In effect, Nimrod transforms file-based programs into interactive “meta-applications” that invoke user programs much as we might call subroutines.

When a user first requests that a computational experiment be performed, Nimrod/G queries MDS to locate suitable resources. It uses information in MDS entries to identify sufficient nodes to perform the experiment. The initial Nimrod-G prototype operates by generating a number of independent jobs, which are then allocated to computational nodes using GRAM. This module hides the nature of the execution mechanism on the underlying platform from Nimrod, hence making it possible to schedule work using a variety of different queue managers without modification to the Nimrod scripts. As a result, a reasonably complex cluster computing system could be retargeted for wide-area execution with relatively little effort.

In the future, the Nimrod-G developers plan to pro-

vide a higher level broker that allows the user to specify time and cost constraints. These constraints will be used to select computational nodes that can meet user requirements for time and cost or, if constraints cannot be met, to explain the nature of the cost/time tradeoffs. As part of this work, a dynamic resource allocation module is planned that will monitor the state of each system and relocate work when necessary in order to meet the deadlines.

6.2 AppLeS

Rich Wolski has used GRAM mechanisms to construct an application-level scheduler (AppLeS) [3] for a large, loosely coupled problem from computational mathematics. As in Nimrod-G, the goal was to map a large number of independent tasks to a dynamically varying pool of available computers. GRAM mechanisms were used to locate resources (including parallel computers) and to initiate and manage computation on those resources. AppLeS itself provided fault tolerance, so that errors reported by GRAM would result in a task being resubmitted elsewhere.

6.3 A Graphical Resource Selector

The graphical resource selector (GRS) illustrated in Figure 7 is an example of an interactive resource selector constructed with our services. This Java application allows the user to build up a network representing the resources required for an application; another network can be constructed to monitor the status of candidate physical resources. A combination of automatic and manual techniques is then used to guide resource selection, eventually generating an RSL specification for the resources in question. MDS services are used to obtain the information used for resource monitoring and selection, and resource co-allocator services are used to generate the GRAM requests required to execute a program once a resource selection is made.

7 Resource Co-allocation

Through the actions of one or more resource brokers, the requirements of an application are refined into a ground RSL expression. If the expression consists of a single resource request, it can be submitted directly to the manager that controls that resource. However, as discussed above, a metacomputing application often requires that several resources—such as two or more computers and intervening networks—be allocated simultaneously. In these cases, a resource broker produces a multirequest, and co-allocation is required. The challenge in responding

to a co-allocation request is to allocate the requested resources in a distributed environment, across two or more resource managers, where global state, such as availability of a set of resources, is difficult to determine.

Within our resource management architecture, multi-requests are handled by an entity called a resource co-allocator. In brief, the role of a co-allocator is to split a request into its constituent components, submit each component to the appropriate resource manager, and then provide a means for manipulating the resulting set of resources as a whole: for example, for monitoring job status or terminating the job. Within these general guidelines, a range of different co-allocation services can be constructed. For example, we can imagine allocators that

- mirror current GRAM semantics: that is, require all resources to be available before the job is allowed to proceed, and fail globally if failure occurs at any resource;
- allocate at least N out of M requested resources and then return; or
- return immediately, but gradually return more resources as they become available.

Each of these services is useful to a class of applications. To date, we have had the most experience with a co-allocator that takes the first of these approaches: that is, extends GRAM semantics to provide for simultaneous allocation of a collection of resources, enabling the distributed collection of processes to be treated as a unit. We discuss this co-allocator in more detail.

Fundamental to a GRAM-style concurrent allocation algorithm is the ability to determine whether the desired set of resources is available at some time in the future. If the underlying local schedulers support reservation, this question can be easily answered by obtaining a list of available time slots from each participating resource manager, and choosing a suitable timeslot [23]. Ideally, this scheme would use transaction-based reservations across a set of resource managers, as provided by Gallop [26]. In the absence of transactions, the ability either to make a tentative reservation or to retract an existing reservation is needed. However, in general, a reservation-based strategy is limited because currently deployed local resource management solutions do not support reservation.

In the absence of reservation, we are forced to use indirect methods to achieve concurrent allocation. These methods optimistically allocate resources in the hope that the desired set will be available at some "reasonable" time in the future. Guided by sources of information, such as the current availability of resources (provided by MDS) or queue-time estimation [24, 7], a resource broker can construct an RSL request that is *likely*, but not guaranteed, to succeed. If for some reason the allocation eventually

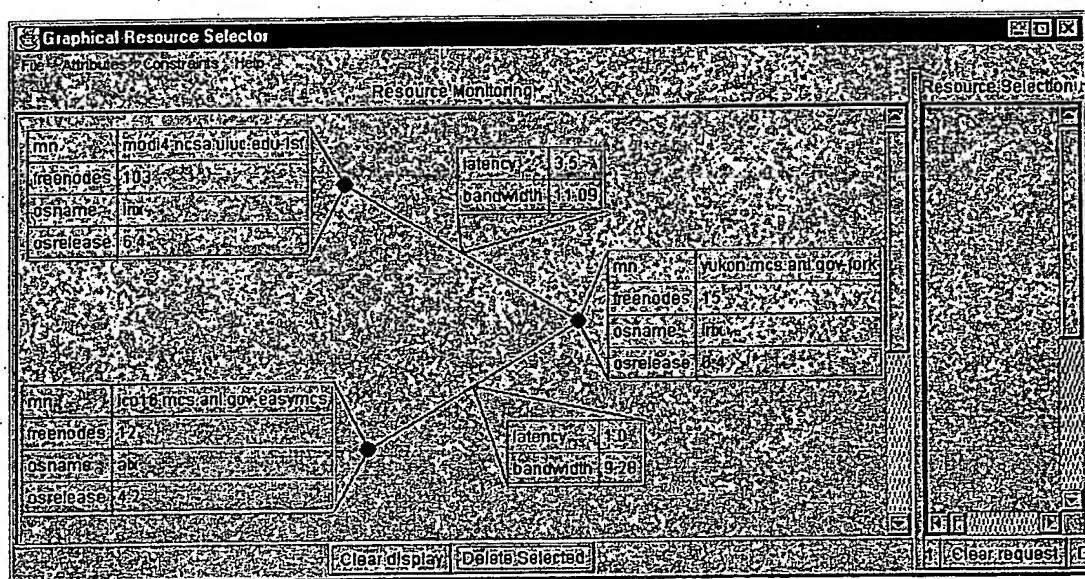


Figure 7: A screen shot of the Graphical Resource Selector. This network shows three candidate resources and associated network connections. Static information regarding operating system version and dynamically updated information regarding the number of currently available nodes (**freenodes**) and network latency and bandwidth (in msec and Mb/s, respectively), allows the user to select appropriate resources for a particular experiment.

fails, all of the started jobs must be terminated. This approach has several drawbacks:

- It is inefficient in that computational resource are wasted while waiting for all of the requested to become available.
- We need to ensure that application components do not start to execute before the co-allocator can determine whether the request will succeed. Therefore, the application must perform a barrier operation to synchronize startup across components, meaning that the application must be altered beyond what is required for GRAM.
- Detecting failure of a request can be difficult if some of the request components are directed to resource managers that interface to queue-based local resource management systems. In these situations, a timeout must be used to detect failure.

However, in spite of all of these drawbacks, co-allocation can frequently be achieved in practice as long as the resource requirements are not large compared with the capacity of the metacomputing system.

We have implemented a GRAM-compatible co-allocator that implements a job abstraction in which multiple GRAM subjobs are collected into a single distributed job entity. State information for the distributed job is synthesized from the individual states of each subjob, and job control (e.g., cancellation) is automatically propagated to the resource managers at each subjob site. Subjobs are started independently and as discussed above must perform a runtime check-in operation. With the exception of this check-in operation, the co-allocator interface is a drop-in replacement for GRAM.

We have used this co-allocator to manage resources for SF-Express [19, 4], a large-scale distributed interactive simulation application. Using our co-allocator and the GUSTO testbed, we were able to simultaneously obtain 852 compute nodes on three different architectures located at six different computer centers, controlled by three different local resource managers. The use of a co-allocation service significantly simplified the process of resource allocation and application startup.

Running SF-Express "at scale" on a realistic testbed allowed us to study the scalability of our co-allocation strategy. One clear lesson learned is that the strict "all or nothing" semantics of the distributed job abstraction severely limits scalability. Even if each individual parallel computer is reasonably reliable and well understood, the probability of subjob failure due to improper configuration, network error, authorization difficulties, and the like, increases rapidly as the number of subjobs increases. Yet many such failure modes resulted simply

from a failure to allocate a specific instance of a commodity resource, for which an equivalent resource could easily have been substituted. Because such failures frequently occur after a large number of subjobs have been successfully allocated, it would be desirable to make the substitution dynamically, rather than to cancel all the allocations and start over.

We plan to extend the current co-allocation structure to support such dynamic job structure modification. By passing information about the nature of the subjob failure out of the co-allocator, a resource broker can edit the specification, effectively implementing a backtracking algorithm for distributed resource allocation. Note that we can encode the necessary information about failure in a modified version of the original RSL request, which can be returned to the component that originally requested the co-allocation services. In this way, we can iterate through the resource-broker/co-allocation components of the resource management architecture until an acceptable collection of resources has been acquired on behalf of the application.

8 Conclusions

We have described a resource management architecture for metacomputing systems that addresses requirements of site autonomy, heterogeneous substrates, policy extensibility, co-allocation, and online control. This architecture has been deployed and applied successfully in a large testbed comprising 15 sites, 330 computers, and 3600 processors, within which LSF, NQE, LoadLeveler, EASY, Fork, and Condor were used as local schedulers.

The primary focus of our future work in this area will be on the development of more sophisticated resource broker and resource co-allocator services within our architecture, and on the extension of our resource management architecture to encompass other resources such as disk and network. We are also interested in the question of how policy information can be encoded so as to facilitate automatic negotiation of policy requirements by resources, users, and processes such as brokers acting as intermediaries.

Acknowledgments

We gratefully acknowledge the contributions made by many colleagues to the development of the GUSTO testbed and the Globus resource management architecture: in particular, Doru Marcușiu at NCSA and Bill Saphir at NERSC. This work was supported by DARPA under contract N66001-96-C-8523, and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technol-

ogy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

- [1] Cray Research, 1997. Document Number IN-2153 2/97.
- [2] D. Abramson, R. Sasic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc. 4th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1995.
- [3] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing '96*. ACM Press, 1996.
- [4] S. Brunett and T. Gottschalk. Scalable ModSAF simulations with more than 50,000 vehicles using multiple scalable parallel processors. In *Proceedings of the Simulation Interoperability Workshop*, 1997.
- [5] S. Chapin. Distributed scheduling support in the presence of autonomy. In *Proc. Heterogeneous Computing Workshop*, pages 22-29, 1995.
- [6] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The Network-Enabled Optimization System (NEOS) Server. Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, Illinois, 1996.
- [7] A. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Parallel Processing Symposium*, 1997.
- [8] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 365-375. IEEE Computer Society Press, 1997.
- [9] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-WAY meta-computing experiment. *Concurrency: Practice & Experience*, 1998. to appear.
- [10] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115-128, 1997.
- [11] GENIAS Software GmbH. CODINE: Computing in distributed networked environments, 1995. <http://www.genias.de/genias/english/codine.html>.
- [12] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Reynolds, Jr. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, Department of Computer Science, University of Virginia, 1994.
- [13] The PSCHED API Working Group. PSCHED: An API for parallel job/resource management version 0.1, 1996. <http://parallel.nas.nasa.gov/PSCHED/>.
- [14] R. Henderson and D. Tweten. Portable Batch System: External reference specification. Technical report, NASA Ames Research Center, 1996.
- [15] International Business Machines Corporation, Kingston, NY. *IBM Load Leveler: User's Guide*, September 1993.
- [16] J. Jones and C. Brickell. Second evaluation of job queuing/scheduling software: Phase 1 report. NAS Technical Report NAS-97-013, NASA Ames Research Center, Moffett Field, CA 94035-1000, 1997. <http://science.nas.nasa.gov/Pubs/TechReports/NASreports/NAS-97-013/jms.eval.rep2.html>.
- [17] David A. Lifka. The ANL/IBM SP scheduling system. In *The IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 187-191, April 1995.
- [18] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proc. 8th Intl Conf. on Distributed Computing Systems*, pages 104-111, 1988.
- [19] P. Messina, S. Brunett, D. Davis, T. Gottschalk, D. Curkendall, L. Ekroot, and H. Siegel. Distributed interactive simulation for synthetic forces. In *Proceedings of the 11th International Parallel Processing Symposium*, 1997.
- [20] K. Moore, G. Fagg, A. Geist, and J. Dongarra. Scalable networked information processing environment (SNIPE). In *Proceedings of Supercomputing '97*, 1997.
- [21] B. C. Neuman. Prospero: A tool for organizing internet resources. *Electronic Networking: Research, Applications, and Policy*, 2(1):30-37, Spring 1992.
- [22] B. C. Neuman and S. Rao. The Prospero resource manager: A scalable framework for processor allocation in distributed systems. *Concurrency: Practice & Experience*, 6(4):339-355, 1994.
- [23] R. Ramamoorthi, A. Rifkin, B. Dimitrov, and K.M. Chandy. A general resource reservation framework for scientific computing. In *Scientific Computing in Object-Oriented Parallel Environments*, pages 283-290. Springer-Verlag, 1997.
- [24] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
- [25] A. Vahdat, P. Eastham, C. Yoshikawa, E. Belani, T. Anderson, D. Culler, and M. Dahlin. WebOS: Operating system services for wide area applications. Technical Report UCB CSD-97-938, U.C. Berkeley, 1997.
- [26] J. Weissman. Gallop: The benefits of wide-area computing for parallel processing. Technical report, University of Texas at San Antonio, 1997.
- [27] J. Weissman and A. Grimshaw. A federated model for scheduling in wide-area systems. In *Proc. 5th IEEE Symp. on High Performance Distributed Computing*, 1996.
- [28] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proc. Workshop on Cluster Computing*, 1992.

FAQ



A range of grid-related questions frequently asked by IBMers and customers alike.

- ↓ [What is a grid?](#)
- ↓ [What effect does grid have on users whose machines are being utilized for processing?](#)
- ↓ [Is grid computing available today — or is it more of a future statement?](#)
- ↓ [What industries are using grid computing now?](#)
- ↓ [What are the possible benefits of a grid deployment?](#)
- ↓ [What is IBM's relationship with grid computing?](#)
- ↓ [Grid and e-business on demand: what's the connection?](#)
- ↓ [Does IBM use grid computing in its own infrastructure?](#)
- ↓ [If I want to learn more about IBM Grid Computing, what's the first step?](#)
- ↓ [What does it take to build a grid?](#)
- ↓ [What about security in grid environments?](#)

What is a grid?

All or some of a group of computers, servers and storage across an enterprise, virtualized as one large computing system. Because grids unleash latent power that, at any one time, is not being used, they can give companies a huge gain in power, speed and collaboration, radically accelerating compute-intensive processes. Cost, meanwhile, can remain low, as grids can be built using existing infrastructure, helping to ensure optimal utilization of computing capabilities.

↑ [Back to top](#)

What effect does grid have on users whose machines are being utilized for processing?

Grids are designed to be seamless and transparent. A user whose desktop PC, say, is contributing processing power to the grid will experience no negative effects: the grid runs in the background, utilizing available resources when needed by the system. If the PC user decides to run an application that requires more processing power, the work currently being processed on that machine will be dynamically reallocated to another machine in the grid with available processing power.

↑ [Back to top](#)

Is grid computing available today — or is it more of a future

statement?

Grid computing is used today by many companies across a number of industries. Current IBM customer references for grid include Butterfly.net, a development studio, online publisher and infrastructure provider for massively multiplayer games that connect players on PC's, consoles and mobile devices. Butterfly Grid consists of two clusters of approximately 50 IBM @server™ xSeries™ servers running in IBM hosting facilities. Specialized game servers and database servers are fully meshed over high-speed fiber-optic lines, enabling transparent routing of players to different servers in the grid. Another current reference for IBM Grid Computing is the University of Pennsylvania's groundbreaking National Digital Mammography Archive, which gives rapid retrieval of digital patient files from multiple locations in a secure environment. The University of Pennsylvania Grid manages this huge data volume, schedules traffic and encrypts all image and information transmission using portal systems running almost exclusively on IBM hardware — including sixteen distributed IBM Netfinity servers running Linux and Windows 2000.

† [Back to top](#)

What industries are using grid computing now?

Some examples include: Automotive and aerospace, for collaborative design and data-intensive testing; financial services, for running long, complex scenarios and arriving at more accurate decisions; life sciences, for analyzing and decoding strings of biological and chemical information; government, for enabling seamless collaboration and agility in both civil and military departments and agencies; higher education for enabling advanced, data and compute intensive research.

† [Back to top](#)

What are the possible benefits of a grid deployment?

Benefits can be extensive. They include:

- Accelerated time to results, which allows for the provisioning of extra time and resources to solve problems that were previously unsolvable
- Improved productivity and collaboration
- Allowing widely dispersed departments and businesses to create virtual organizations to share data and resources
- More flexible, resilient operational infrastructures
- Instantaneous access to compute and data resources to "sense and respond" to needs
- Leveraging existing capital investments, which helps to ensure optimal utilization of computing capabilities
- Avoiding common pitfalls of over-provisioning and incurring excess costs
- Freeing IT organizations from the burden of administering

disparate, non-integrated systems

† [Back to top](#)

What is IBM's relationship with grid computing?

IBM views grid computing as critical to the ongoing development of on demand operating environments. For all the excitement and innovation that grid represents, much of the thinking and technology that drive grid are anything but new to IBM. IBM was an early leader in "virtualization" — the driving force behind grid computing — which has enabled the computer to do many processing jobs simultaneously for thousands of users. Grid computing is an advanced evolution of virtualization — and IBM Grid Computing continues IBM's history of IT innovation for business. Deep experience in e-business processes, support for open standards, enabling of our products and services for grid, partnership role in the grid community and relationships with Business Partners make IBM an important force in bringing the benefits of grid to enterprise computing.

† [Back to top](#)

Grid and e-business on demand: what's the connection?

Grid computing is a key element in e-business on demand. Because it enables new kinds of power, flexibility and integration, IBM Grid Computing is a key element of the on demand operating environment.

† [Back to top](#)

Does IBM use grid computing in its own infrastructure?

Yes. IBM is a major user of grid computing. IBM's intraGrid, based on the Globus Toolkit, is a research and development grid that allows IBM to leverage many worldwide assets for research purposes and help us understand the complexities of managing a grid infrastructure on an enterprise scale. And IBM uses grids for other purposes throughout the company. One example is the IBM Boeblingen Lab Grid, composed of three IBM ~~@server~~ pSeries™ clusters running AIX and LoadLeveler, a cross-departmental grid used to run zSeries processor unit simulations. Jobs are submitted through a web portal, presenting users with the same interface as the one they used when running simulations on an isolated cluster. The WebSphere based portal uses the Globus Java CoG Kit to pre-select candidate queues for submitting each simulation, using Globus Metacomputing Directory Service. This pre-selection is based on cluster loads and job characteristics. Access to a shared DB2 database allows for the automated generation of proxy certificates and for the monitoring and reporting of user jobs.

† [Back to top](#)

If I want to learn more about IBM Grid Computing, what's the

first step?

Sign up for a Grid Innovation Workshop. These sessions offer a hands-on, business-specific understanding of grid computing's strategic, financial and operational advantages for your business. Customized to individual organizations, IBM Grid Innovation Workshops help companies examine how grid technology can help solve their specific information problems. The Workshop includes an Executive Session, work sessions, validation of findings and a preliminary plan.

To sign up, contact us today

† [Back to top](#)

What does it take to build a grid?

Building a grid can be as simple as enabling a small number of PCs (or server or storage network) to take advantage of underutilized processing and storage. This can radically speed completion of a single set of data- or compute-intensive tasks. From a relatively small deployment, you could expand slowly or quickly, narrowly or widely, depending on business needs. Ultimately, an entire enterprise can be enabled for grid — and grids can bring together not only departments and processes within a single company but also those among separate enterprises.

† [Back to top](#)

What about security in grid environments?

Grid Security Infrastructure (GSI) is a public-key-based security protocol, using X.509 certificates, a widely employed standard. The protocol provides single sign-on authentication, which allows a user to create a proxy credential that can authenticate with any remote service on the user's behalf, as well as communication protection and initial support for restricted delegation.

† [Back to top](#)

**NEOS and CONDOR: Solving
Optimization Problems Over the
Internet**

*Michael C. Ferris, Michael P. Mesnier,
and Jorge J. More'*

**CRPC-TR98763-S
March 1998**

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Submitted August 1998; Also available as Argonne
Preprint ANL/MCS-P708-0398

ARGONNE NATIONAL LABORATORY

9700 South Cass Avenue

Argonne, Illinois 60439

**NEOS AND CONDOR: SOLVING OPTIMIZATION PROBLEMS
OVER THE INTERNET**

Michael C. Ferris, Michael P. Mesnier, and Jorge J. Moré

Mathematics and Computer Science Division

Preprint ANL/MCS-P708-0398

March 1998

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by the National Science Foundation under Grants CDA-9726385 and CCR-9619765, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008.

NEOS AND CONDOR: SOLVING OPTIMIZATION PROBLEMS OVER THE INTERNET*

Michael C. Ferris[†], Michael P. Mesnier[‡], Jorge J. Moré[§]

Abstract

We discuss the use of Condor, a distributed resource management system, as a provider of computational resources for NEOS, an environment for solving optimization problems over the Internet. We also describe how problems are submitted and processed by NEOS, and then scheduled and solved by Condor on available (idle) workstations.

1 Introduction

The NEOS Server [8] is a novel environment for solving optimization problems over the Internet. There is no need to download an optimization solver, write code to call the optimization solver, or compute derivatives for nonlinear problems. NEOS provides the user with the input format and a list of solvers for the optimization problem. Given an optimization problem, NEOS solvers compute derivatives and sparsity patterns of nonlinear problems with automatic differentiation tools, link with the appropriate libraries, and execute the resulting binary. The user is provided with a solution and runtime-statistics.

Each solver in the NEOS optimization library is maintained by a software administrator that is responsible for providing computing resources and for answering questions related to the solver. Registering the solver [8] on a few workstations provides adequate resources in most cases, but for large problems, however, we need a different approach. The obvious difficulty is that the owner of a workstation is reluctant to provide large amounts of computing cycles and memory. We use Condor [15, 11], a distributed resource management system, as a provider of computational resources for a NEOS solver. The resources that are managed by Condor are typically large clusters of workstations, many of which would otherwise be idle for long periods of time.

We discuss the connection between NEOS and Condor in the context of a single but important optimization problem: mixed complementarity problems. Our discussion shows that this approach can be extended to other problems as well.

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by the National Science Foundation under Grants CDA-9726385 and CCR-9619765, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008.

[†]Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton St., Madison, Wisconsin 53706. ferris@cs.wisc.edu

[‡]Department of Computer Science, University of Illinois, 1304 W. Springfield, Urbana, Illinois, 61801. mesnier@cs.uiuc.edu

[§]Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439. more@mcs.anl.gov

Many different applications can be formulated as mixed complementarity problems; examples are given in [9, 13]. If the nonlinear function $F: \mathbb{R}^n \mapsto \mathbb{R}^n$ describes the interactions of a nonlinear process as a function of the variables $x \in \mathbb{R}^n$, then the mixed complementarity problem is to find a vector x , with components between lower and upper bounds ℓ and u (with $\ell < u$), such that

$$\begin{aligned} F_i(x) &= 0 & \text{if } \ell_i < x_i < u_i, \\ F_i(x) &\geq 0 & \text{if } x_i = \ell_i, \\ F_i(x) &\leq 0 & \text{if } x_i = u_i. \end{aligned} \tag{1.1}$$

Solving a mixed complementarity problem in the typical computational environment requires that the user first develop code for the evaluation of F . The user must then decide on an appropriate solver, retrieve the solver, develop code to evaluate the Jacobian matrix $F'(x)$ and sparsity pattern of $F'(x)$, link their code with the necessary libraries, and finally execute the solver locally. With NEOS, the user need only specify the mixed complementarity problem by providing code to evaluate the function F , the lower and upper bounds ℓ and u , and a starting point. The NEOS solver then generates code to compute the Jacobian matrix and sparsity pattern, compiles the user subroutines, links with the appropriate libraries, executes the solver on a NEOS machine, and returns a solution to the user.

NEOS uses the Condor pool at the University of Wisconsin for solving complementarity problems. The pairing of NEOS with Condor is an ideal combination. NEOS provides an interface that is problem oriented and independent of the computing resources. Users need only provide a specification of the problem; all other information needed to solve the problem is determined by the NEOS solver. Condor provides the computational resources to solve the problem.

Condor acts as a matchmaker, pairing computational resources with jobs that require processing. The job executes on the allocated workstation until completion or until the workstation becomes unavailable. In the latter case, the job is frozen in its current state and the workstation is returned to the owner. Condor is then contacted once again for pairing and the job is restarted from its frozen state on the newly allocated resource. Condor pays special attention to the needs of the workstation owner by allowing the owner to define the conditions under which the workstation can be allocated. This policy encourages workstation owners to place their resources in the Condor pool, and as a consequence, the Wisconsin pool currently has over 400 workstations.

In Section 2 we describe the three interfaces that are currently available to submit problems to NEOS: e-mail, the NEOS Submission Tool (`neos-submit`), and the NEOS Web interface. These interfaces are designed so that problem submission is intuitive and requires only essential information. Parameters that affect the progress of the solver are not required but can be specified, for example, by an auxiliary file. We concentrate on the NEOS Submission Tool. The NEOS Web interface can be sampled by visiting the URL

for the NEOS Server. We emphasize mixed complementarity problems, but NEOS handles a wide variety of linear and nonlinearly constrained optimization problems; solvers for optimization problems subject to integer variables are being added. We do not discuss the design and implementation of the Server because these issues are covered by Czyzyk, Mesnier, and Moré [8]. Extensions to the NEOS Server and the network computing issues that arise from the emerging style of computing used by NEOS are discussed by Gropp and Moré [14].

Mixed complementarity problems submitted to the NEOS Server are currently solved by the PATH [10, 12] solver, which implements a Newton-type method for solving systems of non-differentiable equations. Sparse matrix techniques are used for large problems. The process used to solve a nonlinear complementarity problem by this NEOS solver includes the generation of derivative and sparsity information with the ADIFOR [4, 5] automatic differentiation tool and the solution of the problem with Condor. The process is governed by a *solver script* that must check the user data and provide appropriate messages in the case of errors. In Section 3 we describe the various issues that must be addressed by the solver script. These issues are important to the development of reliable optimization software and problem-solving environments.

The automatic differentiation techniques used to generate derivatives and sparsity patterns for nonlinear complementarity problems are described in Section 4. In particular, we explain how to obtain a sparse representation of the Jacobian matrix that is suitable for PATH. The Jacobian matrix generated by ADIFOR is accurate to full machine precision, while the Jacobian matrix generated by differences of function values suffers from truncation errors. Moreover, the code produced by ADIFOR for the computation of the sparse Jacobian matrix is typically more efficient than the code produced by differences. On the other hand, the code produced by ADIFOR may not be as efficient as a hand-coded Jacobian matrix. See [1] for a full comparison (in terms of memory and speed) of ADIFOR-generated Jacobian matrices with both hand-coded and difference approximations, and [2] and [7] for performance issues related to the automatic computation of gradients.

The automatic generation of the Jacobian matrix and sparsity pattern in the NEOS version of PATH makes the code more accessible and useful than requiring the hand-coding of the Jacobian matrix. Indeed, all nonlinear solvers in NEOS use automatic differentiation tools to compute gradients, Jacobians, and sparsity patterns. We intend to incorporate ADIC [6] into most of the nonlinear NEOS solvers to allow problems to be specified in C, as well as Fortran.

The final section of the paper describes how the Condor system at the University of Wisconsin is used to process the submitted jobs. Only large jobs are scheduled on Condor because there may be a delay in execution while waiting for an idle workstation. Small

jobs are executed immediately on non-clustered workstations. No computational results are given here. Users are encouraged either to submit one of the supplied sample problems or to generate new mixed complementarity problems to test the system.

2 The NEOS Server

The NEOS Server provides Internet access to a library of optimization solvers with user interfaces that abstract the user from the details of the optimization software. The user needs only to describe the optimization problem in a particular format; all additional information required by the optimization solver is determined automatically. This abstraction is similar to that provided by modeling languages. The NEOS solvers provide several different input formats to allow users to specify optimization problems in a convenient manner, without necessarily rewriting their problem in a modeling language.

The NEOS approach offers considerable advantages over a conventional environment for solving optimization problems. Consider, for example, mixed complementarity problems. A NEOS solver for mixed complementarity problems requires that the user specify the number of variables n , a subroutine `initpt(n,x)` that defines the starting point, a subroutine `xbound(n,xl,xu)` that sets the lower and upper bounds, and a subroutine `fcn(n,x,f)` that evaluates the function F . Since there is no need to provide the Jacobian matrix or the sparsity pattern of the Jacobian matrix, the user can concentrate on the specification of the problem. Changes to the `fcn` subroutine can be made and tested immediately; the advantages in terms of ease of use are considerable.

Other optimization problems can be specified in a similar manner. For example, the nonlinearly constrained optimization problem

$$\min \{f(x) : x_l \leq x \leq x_u, c_l \leq c(x) \leq c_u\}$$

can be specified by four subroutines. The bounds x_l and x_u are specified with the subroutine `xbound(n,xl,xu)`, the constraint bounds c_l and c_u are specified with the subroutine `cbound(m,cl,cu)`, the objective function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is defined by the subroutine `fcn(n,x,f)`, and the nonlinear function $c : \mathbb{R}^n \mapsto \mathbb{R}^m$ is defined by `cfcn(m,x,c)`.

We have mentioned nonlinear optimization solvers, but NEOS contains solvers in other areas. A complete listing is available at the NEOS Server homepage:

<http://www.mcs.anl.gov/otc/Server/>

The addition of solvers is not difficult. Indeed, as discussed in [8], NEOS was designed so that solvers in a wide variety of optimization areas can be added easily.

We provide Internet users the choice of three interfaces for submitting problems: e-mail, the NEOS Submission Tool, and the NEOS Web interface. These interfaces are designed so that problem submission is intuitive and requires the minimal amount of information.

Figure 2.1: The NEOS submission form for PATH

The interfaces differ only in the way that information is specified and passed to the NEOS Server.

The e-mail interface is relatively primitive, but useful because most users have easy access to e-mail. Information on the available solvers and on the format used to submit problems via e-mail can be obtained by sending the mail message help to

neos@mcs.anl.gov

Users interested in the Web interface should visit the homepage for the NEOS Server, which has links to all the solvers in the library, as well as pointers to other NEOS information, in particular, the NEOS Guide. In the remainder of this section we examine the NEOS Submission Tool.

The NEOS Submission Tool provides a high-speed link to the NEOS Server via TCP/IP sockets. Once this tool is installed (only Perl [17] is required), the user has access to all solvers offered by NEOS. Additional information on the NEOS Submission Tool, including installation instructions, can be obtained from the NEOS Server homepage.

Submission of problems via the NEOS Submission Tool is simple. The user must first choose the type of optimization problem and then select the desired solver. Once the solver is selected, the user is given a submission form specific to the solver.

The PATH submission form, shown in Figure 2.1, requires that the user specify the number of variables, the files for the initial point, bounds on the variables, and function evaluation subroutines.

Figure 2.1 shows the NEOS Submission form for a model of oligopolistic pricing [16]

with 63 variables. The model is defined by the file `op.fcn`, while the initial point and the bounds on the problem are defined by the files `op_initpt` and `op_xbound`. Note that in this case two PATH options are set and that we have requested the use of Condor for the solution of this problem.

The user has the option of using a Condor pool of workstations for solving the submitted problem. Since Condor is essentially a batch processing mechanism, the user is also allowed to specify a timeout for Condor. After this period of time, the Web browser or Submission Tool is released from its busy state and returned to the user. The job, however, continues to process, and the results are returned to the user at a later date via e-mail. The default timeout is 5 minutes; Condor is used by default on all problems that are larger than 500 variables.

The PATH submission form allows the user to provide a specification file that can be used to set tolerances and other parameters that govern the algorithm. For most problems the defaults provided are adequate. Figure 2.1 shows two options in use for this submission. The first provides a listing of the current settings of all the available options for this run; the second just turns off the default crash technique. The form also has room for comments, which can be used to identify the problem submission.

Once specified, the problem is submitted to NEOS where it is then scheduled for execution. A variety of computers, even a massively parallel processor, could be used to solve the problem. At present these computers are workstations that reside at Argonne National Laboratory, Northwestern University, the University of Wisconsin, Lawrence Berkeley National Laboratory, the Technical University of Ilmenau in Germany, and Arizona State University.

3 Solving Complementarity Problems: PATH

The process used to solve a nonlinear complementarity problem by NEOS is illustrated in Figure 3.1. This process includes the generation of derivative and sparsity information with the ADIFOR [5, 4] automatic differentiation tool and the solution of the problem with the Condor [15, 11] distributed resource management system. We discuss ADIFOR further in Section 4, while Condor is discussed in Section 5. In this section we discuss issues in the solution process that are pertinent to the development of optimization software and problem-solving environments. Although the discussion is specific to PATH [10, 12], most of the issues are applicable to all the solvers of nonlinear optimization problems in NEOS.

Submitting a problem to the NEOS Server does not guarantee success, but NEOS users are able to solve difficult optimization problems without worrying about many of the details that are typical in a conventional computing environment. Even if the user has suitable optimization software, the user would need to read the documentation, write code to interface his problem with the optimization software, and then debug this code. The user would also have to write code to evaluate the Jacobian matrix and sparsity pattern, and debug that

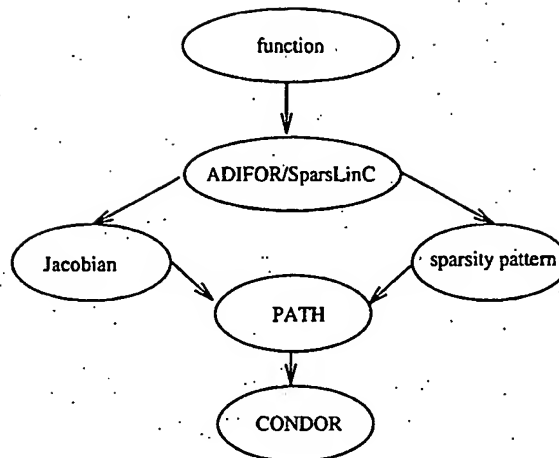


Figure 3.1: PATH and Condor

code—a nontrivial undertaking in most cases.

For a typical submission, the user receives information on the progress, and the solution. Figure 3.2 shows part of the output received when the problem in Figure 2.1 is submitted to NEOS via the NEOS Submission Tool. In particular, we see the NEOS server selecting an available workstation, transferring all user data to the workstation, and then invoking the solver remotely. The solver (in this case PATH) checks the data and compiles the user's code. If any errors are found at this stage, the compiler error messages are returned to the user, and execution terminates.

If the user's code compiles correctly, the automatic differentiation tool ADIFOR [5, 4] is used to generate the Jacobian matrix and the sparsity pattern. Additional details on this part of the process are discussed in Section 4. Once the Jacobian matrix and sparsity pattern are obtained, the user's code is linked with the optimization libraries, and execution begins. Results are returned in the window generated by the NEOS Submission Tool.

The solver script that handles the solution process must check the input data to make sure that the job submission is valid. A typical error at this stage of the solution process is for the user to interchange files and to send, for example, subroutine `initpt` where NEOS is expecting subroutine `xbound`. A similar error is to neglect to send one of the files required for the job submission. These errors are detected by the solver script by checking that the files that specify `fcn`, `initpt`, and `xbound` exist and that they reference the appropriate subroutine. The solver script also checks that the problem dimensions are positive.

Even if the supplied subroutines compile correctly, ADIFOR may find an error during the generation of the `fcn` subroutine. The most common error here is for the submission to contain an improper calling sequence. For example, if the calling sequence of the submitted `fcn` is `fcn(n,x,y)`, ADIFOR generates an error because it is assuming that the independent

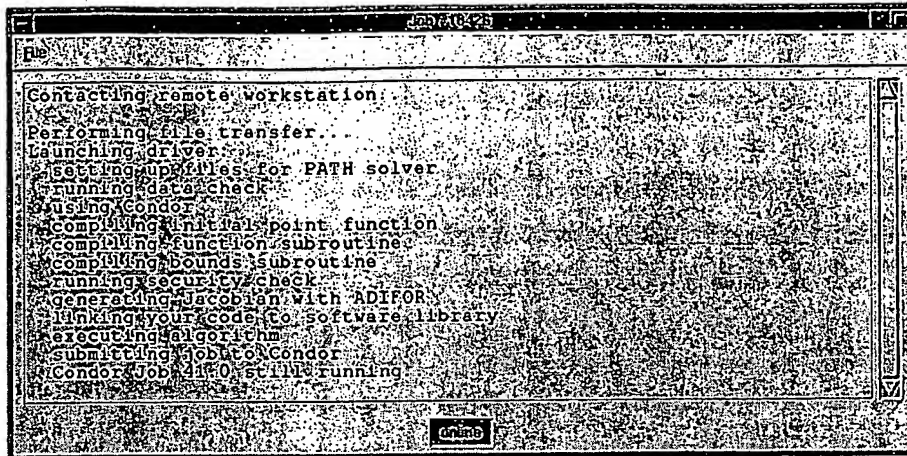


Figure 3.2: Output from the NEOS Submission tool

variables are f . On the other hand, ADIFOR does not generate an error if the calling sequence is $fcn(n, x, f, w)$ because now there is a dependence on f . All error messages generated by ADIFOR are sent back to the user.

If the derivative and sparsity information is generated, this information is sent to PATH. Errors may also occur during this part of the solution process, and it is again important to send appropriate messages back to the user. At present, we check only that the user function does not create any system exceptions during the evaluation of the function at the starting point or at any of the iterates. Although simple, this test catches many user errors. In particular, this test does not allow a calling sequence of the form $fcn(n, x, f, w)$.

Additional checks on the function would be desirable, but seem to be difficult to implement. For example, we would like to check that the function provided is indeed differentiable. If the user provides a function that is discontinuous, automatic differentiation tools will generate the Jacobian matrix but will not be able to detect this situation.

4 ADIFOR

Figure 3.1 shows that given the function F that defines the mixed complementarity problem, the automatic differentiator tool ADIFOR/SparsLinc [3, 5] is used to produce the Jacobian matrix of F and the sparsity structure of the Jacobian matrix. This information is then fed to PATH. In this section we describe the process for generating a representation of the sparse Jacobian matrix of the function F that is submitted to the PATH solver. This process is of interest to any researcher who wishes to use automatic differentiation tools.

The first step in using ADIFOR is to create a *script* file that defines the dependent and independent variables, the name of the subroutine that needs to be differentiated, and a *composition* file.

```

AD_PROG      = fcn.comp
AD_TOP       = fcn
AD_IVARS     = x
AD_DVARS     = f
AD_SEP       = -
AD_OUTPUT_DIR = .
AD_FLAVOR    = sparse

```

Figure 4.1: Script file `fcn.script` for ADIFOR

Figure 4.1 shows the script file that is used with PATH. This file tells ADIFOR that the subroutine that needs to be differentiated is called `fcn`, that the independent variables are `x`, and that the dependent variables are `f`. The composition file is specified with `AD_PROG`, so in this case the composition file is `fcn.comp`.

The composition file contains a list of all the files that are required to compute the function, and also a sample program that specifies the calling sequence for the subroutine `fcn`. The composition file used with PATH is shown below:

```

fcn.f
fcn_sample.f

```

This file tells ADIFOR that file `fcn.f` contains the subroutine `fcn` and that the sample program is contained in file `fcn_sample.f`.

All subroutines that are required to evaluate the function must be in the file `fcn.f`. If other subroutines are needed (for example, some blas subroutines), they also must be included in `fcn.f`.

The sample file `fcn_sample.f` is not strictly needed because we already know that the subroutine to be differentiated is called `fcn`. However, in older versions of ADIFOR, this file is needed. Figure 4.2 shows the sample file that is used with PATH. The only information specified by this file is the calling sequence used by `fcn`.

```

program fcn_sample
integer n
double precision x(n), f(n)
call fcn(n,x,f)
end

```

Figure 4.2: Sample file `fcn.comp` for ADIFOR

Given the information in the script and composition files, ADIFOR can be used to generate a subroutine that computes the Jacobian matrix. Since we are interested in computing *sparse* Jacobian matrices, the subroutine that ADIFOR generates uses special data structures (called *objects*) to define the Jacobian matrix.

The command `Adifor2.0 AD_SCRIPT=fcn.script` instructs ADIFOR to generate a subroutine of the form

```
g_fcn(n,x,g_x,f,g_f)
```

where `g_x` is a gradient object for the independent variable `x` and `g_f` is a gradient object for the function `F`. These objects are manipulated and accessed by `PATH` as described below. Further details on how to invoke ADIFOR can be found in [3, 5].

We compute the Jacobian of `F` by manipulating the gradient object with subroutines provided by ADIFOR in the `SparsLinc` [4] library. We first set the gradient object `g_x` for the independent variable `x` to the identity matrix with the code segment

```
do j = 1, n
  call dspds(g_x(j),j,1.d0,1)
end do
```

Once `g_x` is defined, we use the ADIFOR-supplied subroutine `dspxsq` to compute the Jacobian matrix. The call

```
call dspxsq(ind_col,val,n,g_f(i),lenrow,info)
```

extracts the `i`th row of the Jacobian matrix. On exit from this call to `dspxsq`, the array `ind_col` contains the column indices of the `i`th row of the Jacobian matrix, the array `val` contains the values of the `i`th row, and the variable `lenrow` is the number of nonzeros.

Two key difficulties arise when using the derivative information provided by ADIFOR in `PATH`. The first difficulty is that `PATH`, like most optimization software for sparse problems, assumes that the sparsity structure is known for all values of the independent variables `x`. This information is needed in order to preallocate enough storage for the Jacobian matrix and to minimize the cost of preprocessing the Jacobian matrix. For example, orderings that reduce the fill in an elimination algorithm use the sparsity structure, so if the sparsity structure changes at each iteration, then the ordering will have to be recomputed at each iteration. Dynamic storage allocation schemes could be used, but these schemes tend to increase the overall computing time significantly.

We determine a sparsity structure that is valid for all values of the independent variables `x` by evaluating the Jacobian matrix at a random perturbation of the initial point provided by the user. We cannot use the initial point provided by the user to determine a sparsity structure that is valid for all values of the independent variables `x` because the initial starting point tends to be special (for example, the vector of all zeros or all ones), and thus the resulting sparsity structure is not representative. This heuristic was also used by Bouaricha and Moré [7] in a similar situation.

If the sparsity pattern changes as the iteration proceeds then the heuristic that we are using may fail. However, this situation seems to be rare. Heuristics that detect changes in sparsity patterns are the subject of current research.

The second difficulty is that PATH uses a pivotal method to compute the step between iterates, and this method requires that the Jacobian matrix be stored by columns. On the other hand, ADIFOR computes the Jacobian matrix by rows. Storing the Jacobian matrix in a compressed column format specified by an array `ind_row` of row indices and an array `col_ptr` that points to the start of each column is not difficult. We use an additional array `col_start` that initially agrees with `col_ptr`. As we run through the rows of the Jacobian matrix generated by using `dspxsq`, the entries from `val` are immediately put into their correct location (as determined by `ind_row`), and the corresponding entry of `col_start` is incremented. Note that the resulting column-wise storage is sorted by row indices, even though this is not required by PATH. A final check to determine whether the allocated storage is sufficient is carried out after all rows have been processed.

5 Condor

Condor [11, 15] is a distributed resource management system, developed at the University of Wisconsin, that manages large heterogeneous clusters of workstations. Due to the ever decreasing cost of low-end workstations, such resources are becoming prevalent in many workplaces. The Condor design was motivated by the needs of users who would like to take advantage of the underutilized capacity of these clusters for their long-running, computationally-intensive jobs. Condor has been ported to most UNIX platforms and has been used in production mode for more than eight years in the Computer Sciences Department of the University of Wisconsin and many other sites. A version that runs under Windows NT is under development. The system is publicly available under the GNU copy-left restrictions and can be downloaded from

<http://www.cs.wisc.edu/condor/>

In order to generate vast amounts of computational resources, such a system must use any kind of resource whenever it is made available. Condor acts like a matchmaker, pairing these computational resources with jobs that require processing. The job executes on the allocated machine until it completes or the resource disappears. In the latter case, the job is checkpointed, the machine is returned to the owner, and Condor is contacted once again for pairing. Checkpointing a job is the process of saving the current state of the job in a way that allows restarting from precisely the same point of execution.

Condor preserves a large measure of the originating machine's environment on the execution machine, even if the originating and execution machines do not share a common file system. Condor jobs that consist of a single process (it is possible to run PVM on a Condor cluster) are automatically checkpointed and migrated between workstations as needed to ensure eventual completion. Condor is flexible and fault-tolerant: the design features ensure the eventual completion of the job. This feature is important for our application.

A key design feature of the Condor system is that the owner of the resource should have as little interference from the resource allocation server as possible; in this way, more owners will make their resources available to the pool. Condor pays special attention to the needs of the interactive user of the workstation by allowing the user to define the conditions under which the workstation can be allocated by Condor to a batch user. As a consequence, there are currently over 400 workstations in the Wisconsin pool.

The use of Condor for solving complementarity problems generated from NEOS is intended to be an example, showing how a wide variety of software tools can be interfaced and used in a practical operations research environment. The development of software tools is extremely important, but frequently there are few examples demonstrating how the developer envisioned these tools being used. Such examples serve as a prototype for new applications and show potential users how to develop their applications and problems for network solution.

Figure 3.1 shows all the steps used by NEOS to solve a mixed nonlinear complementarity problem. We have already discussed the generation of derivative and sparsity patterns in Section 4. We now outline how Condor schedules job submissions on an appropriate workstation from the Wisconsin pool.

Using Condor-managed resources is easy. Fortran or C code that runs under one of the supported systems can be relinked by using libraries from the Condor system without any changes to the source code. The solver script schedules only large jobs for this facility, since there may be a delay in execution while waiting for an appropriate idle workstation. Small jobs are executed directly on a non-clustered machine at Wisconsin.

The first step in preparing a code for solution using Condor is to ensure that the code compiles and runs on the native machine of that class. Since the PATH solver is already tested and available in library form, this amounts to checking that the submitted routines and ADIFOR-generated routines can be compiled, exactly as is done for a submission that is to run on a local machine. The second step links these objects to the PATH objects, replacing some of the standard libraries with Condor-supplied replacements. Our interface replaces a single line in the standalone makefile, namely,

```
f77 -o pathsol $(OBJECTS) $(LIBS)
```

with the following line

```
condor_compile f77 -o pathsol $(OBJECTS) $(LIBS)
```

Both of these makefiles use precisely the same library of routines that implement PATH as described in [12]. The only difference is that different system libraries are linked into the executable in order to facilitate the checkpointing that was mentioned above.

Instead of executing `pathsol` on the machine where it was compiled, the solver script generates a *job description file* that details the location of the executable, the requirements of the job, and all input and output files. For NEOS, the job description file `jdf` is

```
Executable = pathsol
Log = condor_dir/condor.log
Coresize = 0
Notification = never
Queue
```

This file specifies, in particular, that `condor.log`, in directory `condor_dir`, is the Condor log file. The purpose of the log file is discussed below.

The job is submitted to Condor by using the command `condor_submit -v jdf`. At this stage, Condor takes over control of the job. For security purposes, the job runs as user `nobody`, thereby limiting the access of the submitted job to files that it owns.

In the remainder of this section, we outline how we allow NEOS to timeout from the Condor job and how we guarantee that the job results are returned to the NEOS user - even if the machine that submitted the job to Condor dies during the execution of the job, or communication between NEOS and Condor dies. Note that the Condor job is detached from the submitting machine and is guaranteed to continue to execute to completion.

We first create a persistent directory, `condor_dir`, on the machine that submits the Condor job. All files related to Condor jobs are located in `condor_dir`. When a job is submitted, we create a symbolic link into `condor_dir` the job directory created by the NEOS Communications Package - the facility enabling communication between a solver and the NEOS Server. This job directory serves as the repository for both incoming job submissions and outgoing results.

The Condor log file, `condor.log`, resides in `condor_dir` and shows where each job from NEOS was submitted and executed. We have created a program for monitoring this log with the `UserLogAPI` that is part of the Condor distribution. The monitoring program, `watchlog`, is regularly invoked by the system utility `cron` and immediately exits if it finds another version of `watchlog` running.

The purpose of `watchlog` is to ensure that the results of any job submitted to Condor are written to the appropriate job directory and to signal that the Condor job is finished. Condor writes to `condor.log` every time the status of the job changes. The `watchlog` monitoring program acts upon two events that are written into `condor.log`, namely, `EXECUTE` and `TERMINATE`. If the job in question starts executing on a machine, `watchlog` adds the IP address of this machine to the list of machines that have been used for this job. At the end of processing, this list is appended to the job results; an example of such a list is given below:

<128.105.40.8:32776>
<128.105.41.104:32839>
<128.105.76.12:36651>
<128.105.5.11:56558>

The second event that triggers the watchlog program is the fact that the job in question is terminating. In this case, watchlog writes the status of the Condor job to a file called CONDOR_DONE in the job directory. Once the job completes, the results are returned to the NEOS user by the mechanism we now describe.

First note that even if a job was executed under Condor, the solver creates exactly the same solution files and writes these files in the job directory as before. To guarantee that the results are returned to the NEOS user we have to deal with two cases. Firstly, the submitting machine may die while the job is being executed under Condor and secondly, the NEOS user may not be willing to wait more than 5 minutes for the job results to be returned. To deal with both these cases, we have created another cron program, job-checker, to ensure that the job results are returned either to NEOS or directly to the user via e-mail.

This program simply monitors the job directory, checking for the existence of the files CONDOR and CONDOR_DONE (signifying that Condor was used and that Condor had completed the job) and that the file DONE does not exist. The solver script creates the file DONE once the user has been notified of the job results. Thus if this file is not present, and the job was a Condor job that had timed-out, we return the job results to the user via e-mail. There is a slight race condition here: it is possible that a (Condor) timeout can occur while the job is being returned to NEOS. In this case, the user may get notified of the results both in the NEOS submission tool or WEB browser and via email. We believe this strategy is preferable to the possibility of losing some results.

Acknowledgments

We thank the developers of Condor and ADIFOR for sharing their expertise with us. Todd Munson deserves special thanks for his contributions to the development of the current version of PATH.

References

- [1] B. M. AVERICK, J. J. MORÉ, C. H. BISCHOF, A. CARLE, AND A. GRIEWANK, *Computing large sparse Jacobian matrices using automatic differentiation*, SIAM J. Sci. Statist. Comput., 15. (1994), pp. 285-294.
- [2] C. BISCHOF, A. BOUARICHA, P. KHADEMI, AND J. J. MORÉ, *Computing gradients in large-scale optimization using automatic differentiation*, INFORMS J. Computing, 9 (1997), pp. 185-194.

- [3] C. BISCHOF, A. CARLE, G. CORLISS, A. GRIEWANK, AND P. HOVLAND, *ADIFOR: Generating derivative codes from Fortran programs*, Scientific Programming, 1 (1992), pp. 1-29.
- [4] C. BISCHOF, A. CARLE, AND P. KHADEMI, *Fortran 77 interface specification to the SparsLinC library*, Technical Report ANL/MCS-TM-196, Argonne National Laboratory, Argonne, Illinois, 1994.
- [5] C. BISCHOF, A. CARLE, P. KHADEMI, AND A. MAUER, *The ADIFOR 2.0 system for the automatic differentiation of Fortran 77 programs*, Preprint MCS-P381-1194, Argonne National Laboratory, Argonne, Illinois, 1994. Also available as CRPC-TR94491, Center for Research on Parallel Computation, Rice University.
- [6] C. BISCHOF, L. ROH, AND A. MAUER, *ADIC: An extensible automatic differentiation tool for ANSI-C*, Software — Practice and Experience, 27 (1997), pp. 1427-1456.
- [7] A. BOUARICHA AND J. J. MORÉ, *Impact of partial separability on large-scale optimization*, Comp. Optim. Appl., 7 (1997), pp. 27-40.
- [8] J. CZYZYK, M. P. MESNIER, AND J. J. MORÉ, *The Network-Enabled Optimization System (NEOS) Server*, Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, Illinois, 1996. To appear in IEEE Computational Science & Engineering.
- [9] S. P. DIRKSE AND M. C. FERRIS, *MCPLIB: A collection of nonlinear mixed complementarity problems*, Optim. Methods Software, 5 (1995), pp. 319-345.
- [10] —, *The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems*, Optim. Methods Software, 5 (1995), pp. 123-156.
- [11] D. H. EPEMA, M. LIVNY, R. VAN DANTZIG, X. EVERS, AND J. PRUYNE, *A world-wide flock of condors: Load sharing among workstation clusters*, Journal on Future Generations of Computer Systems, (1996).
- [12] M. C. FERRIS AND T. S. MUNSON, *Interfaces to PATH 3.0: Design, implementation and usage*, Mathematical Programming Technical Report 97-12, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1997.
- [13] M. C. FERRIS AND J.-S. PANG, *Engineering and economic applications of complementarity problems*, SIAM Rev., 39 (1997), pp. 669-713.
- [14] W. GROPP AND J. J. MORÉ, *Optimization environments and the NEOS server*, in Approximation Theory and Optimization, M. D. Buhmann and A. Iserles, eds., Cambridge University Press, 1997, pp. 167-182.

- [15] M. J. LITZKOW, M. LIVNY, AND M. W. MUTKA, *Condor - A hunter of idle workstations*, in Proceedings of the 8th International Conference on Distributed Computing Systems, Washington, District of Columbia, 1988, IEEE Computer Society Press, pp. 108-111.
- [16] E. SIMANTIRAKI AND D. F. SHANNO, *An infeasible-interior-point algorithm for solving mixed complementarity problems*, in Complementarity and Variational Problems: State of the Art, M. C. Ferris and J. S. Pang, eds., Philadelphia, Pennsylvania, 1997, SIAM Publications, pp. 386-404.
- [17] L. WALL, T. CHRISTIANSEN, AND R. L. SCHWARTZ, *Programming Perl*, O'Reilly & Associates, Inc., second ed., 1996.

A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems

Ian Foster

Mathematics and Computer Science
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439

Nicholas T. Karonis

High-Performance Computing Lab
Department of Computer Science
Northern Illinois University
DeKalb, IL 60115

Abstract

Application development for high-performance distributed computing systems, or computational grids as they are sometimes called, requires "grid-enabled" tools that hide mundane aspects of the heterogeneous grid environment without compromising performance. As part of an investigation of these issues, we have developed MPICH-G, a grid-enabled implementation of the Message Passing Interface (MPI) that allows a user to run MPI programs across multiple computers at different sites using the same commands that would be used on a parallel computer. This library extends the Argonne MPICH implementation of MPI to use services provided by the Globus grid toolkit. In this paper, we describe the MPICH-G implementation and present preliminary performance results.

1 Introduction

High-performance "computational grids" [11] involve heterogeneous collections of computers that may reside in different administrative domains, run different software, be subject to different access control policies, and be connected by networks with widely varying performance characteristics. We believe that application development in these environments requires specialized "grid-enabled" tools that hide mundane aspects of the heterogeneous grid environment without compromising performance. These tools may implement familiar programming models, such as message passing, data parallelism, or object parallelism (perhaps with extensions), or may implement completely new programming models. In either case, research is required to understand the utility of different approaches and the techniques

that may be used to implement these approaches in different environments.

As part of an investigation of these issues, we have developed MPICH-G, a grid-enabled implementation of the Message Passing Interface (MPI) that allows the user to run MPI programs across multiple computers at different sites using the same commands that would be used on a parallel computer. This library extends the Argonne MPICH implementation of MPI [15] to use services provided by the Globus grid toolkit [10], as follows:

1. The Globus information service is used to determine how to obtain access to the computers in question.
2. The Globus security service is used to handle authentication and authorization at each site.
3. The Globus executable management service is used to stage executables.
4. The Globus resource management service is used to start processes on each computer, interfacing with local schedulers where necessary.
5. The Globus communication service is used to manage the different communication methods that may apply in a heterogeneous environment, such as vendor-supplied protocols or TCP/IP.
6. The Globus file access service is used to direct standard output and error (stdout and stderr) streams to the user's terminal and to provide access to files regardless of location.
7. Globus process management facilities allow the programmer to monitor the progress of an application and terminate it if desired.

MPICH-G is a complete implementation of the MPI-1 standard and passes the MPICH test suite. Early experiences suggest that it achieves our goal of reducing barriers to the use of distributed computing by allowing the use of MPI as a portable, high-performance programming model for heterogeneous clusters and for wide-area computing systems. Several groups (e.g., at Lawrence Livermore National Laboratory (LLNL) and NASA Ames Research Center) are using it to run conventional MPI programs across multiple massively parallel processors (MPPs) within the same machine room. In this case, MPICH-G is used primarily to manage startup and to achieve efficient communication via use of different low-level communication methods. Other groups are using MPICH-G for metacomputing experiments, in which applications are distributed across MPPs located at different sites: Larsson for studies of distributed execution of a large computational electromagnetics code [17], and Chen and Taylor in studies of automatic partitioning techniques as applied to finite element codes [4]. MPICH-G can also be used to implement distributed visualization pipelines and similar applications in which components are located at different sites. In these latter examples, MPICH-G is used to manage heterogeneous authentication and startup mechanisms.

In the rest of this article, we describe the problems that we faced in developing MPICH-G, the techniques used to overcome these problems, and preliminary experimental results that indicate the costs associated with the MPICH-G implementation.

2 The Need for Grid-Enabled Tools

An extensive body of experience shows that the coupling of geographically distributed computers, databases, scientific instruments, and people can enable interesting new applications. Distributed supercomputing [19], knowledge synthesis [20], online instrument control [16], and teleimmersion [6] are just four examples. However, experience also shows that the barriers to the construction of such applications are considerable. Few programmers take the time to master the intricacies of such grid environments, and even then often produce applications that are fragile, nonportable, and perform poorly.

The specific problems encountered by the developers of such grid applications vary widely according to the grid environment and application type in question. We use Figure 1 to illustrate some of the problems that we have been concerned with in the development of MPICH-G. This figure shows three massively parallel processing (MPP) systems, each constructed from sym-

metric multiprocessor (SMP) nodes. Two of the MPPs are located within the same institution and hence are connected by some form of (hopefully high-speed) local area network (LAN), while the third is located at a remote site and hence is reached by a wide area network (WAN). The following is a partial list of the problems that we may encounter in such an environment:

1. The two sites will likely operate different authentication and authorization mechanisms and impose different access control policies. A user is unlikely to have the same user id at the two sites.
2. The two sites are unlikely to share a file system. Hence, specialized techniques are required to transfer executables and program files between sites.
3. The different MPPs may be controlled by different schedulers with different scheduling policies.
4. We need to allocate resources concurrently at multiple sites and establish a single computational environment (in MPI terms, a single `MPI_COMM_WORLD`) that spans those resources. (We refer to this as the "co-allocation" problem.)
5. Efficient communication requires that different communication methods be used in different situations. Within an SMP, shared-memory communication should be used, whether by using explicit shared-memory operations or by using shared memory operations to provide fast implementations of other abstractions such as message passing. Between SMPs within the same MPP, a vendor-supplied message-passing library should be used. Only between MPPs should the universally available but slow TCP/IP be used. (An exception to this rule is shown in the upper MPP in Figure 1. In some cases, a limitation on the number of nodes that can communicate using the vendor-supplied library may require the use of TCP/IP even within an MPP.)
6. The topology of the overall computational system needs to be taken into account when implementing communication algorithms. Taking into account the different TCP/IP performance (in terms of both absolute speeds and bisection bandwidths) within an MPP, over a LAN, and over a WAN, the example system features five different communication speed regimes.

We believe that the solution to these types of problem is to develop grid-enabled tools that provide efficient implementations of familiar (or unfamiliar) programming models for use by application developers. In

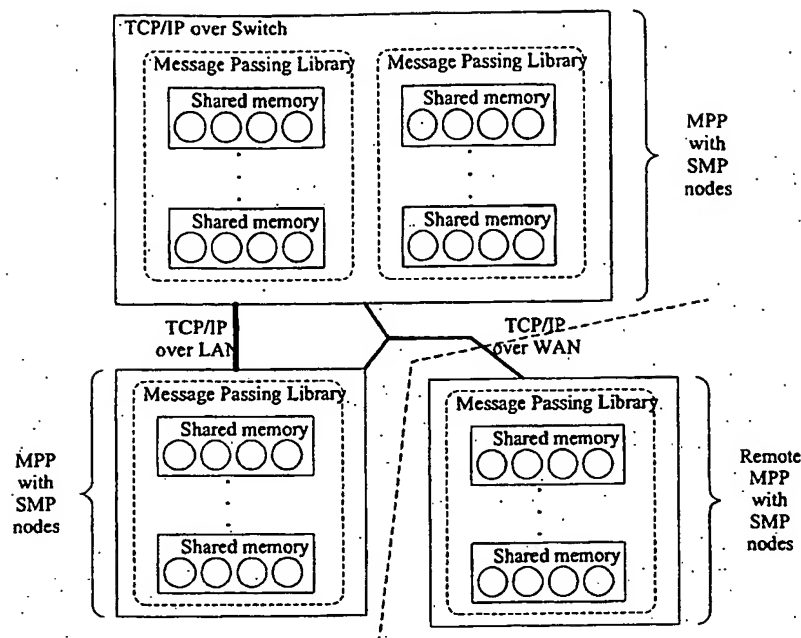


Figure 1. The structure of a prototypical “computational grid” computing environment, of the type supported by MPICH-G. See text for details.

developing these implementations, the tool developer must be concerned not only with translating the programming model to the grid environment, but also with revealing to the programmer those aspects of the grid environment that impact performance. For example, a grid-enabled MPI might handle automatically issues of authorization, startup, and process management, hence addressing the first four points listed above. It might also incorporate specialized techniques for point-to-point and collective communication in highly heterogeneous environments, hence addressing points 5 and 6. Finally, it might also extend the MPI model to provide programmers with access to resource location services, information about grid topology, group communication protocols, and quality-of-service management services, so as to enable new programming techniques appropriate for grid environments.

In principle, such grid-enabled tools could be constructed from scratch. However, the task is greatly simplified if the programmer has access to appropriate low-level services. As we explain below, we use the Globus toolkit as a source of such services in our work.

The state of the art with respect to such tools is not very advanced. Systems such as Condor [18], NEOS [5], and NetSolve [3] all implement grid-based programming models of various sorts. Various implementations

of message passing libraries provide some support for heterogeneous execution (e.g., p4 [2] and PVM [14]), but these systems do not support the flexible use of alternative low-level communication protocols, interfaces to different MPP schedulers, or the MPI standard. PVMPI [7] exploits a renaming capability provided by MPI’s profiling interface to use PVM mechanisms to couple vendor-supplied MPIs on different MPPs. The resulting system supports heterogeneous execution of MPI programs but cannot deal with heterogeneous startup mechanisms or dynamic selection of communication methods.

3 Building Blocks

Our grid-enabled MPI implementation is constructed from two existing software systems: MPICH and Globus. We describe these briefly here.

3.1 MPICH

MPICH [15] is the most widely used implementation of the MPI standard. Its architecture features a layered design, in which higher-level MPI communication constructs such as collective operations, communicators, and topologies are implemented in terms of ba-

sic communication operations provided by an "abstract device." Various such devices have been designed, enabling high-performance implementations of MPICH on a variety of platforms. We exploit this device architecture in our work, defining a "Globus communication device" that supports the use of multiple low-level communication methods in heterogeneous wide area environments.

MPICH also defines a uniform startup mechanism for MPI programs. For example, the command

```
mpirun -np 64 myprog
```

starts the MPI program `myprog` as 64 processes, whether on a shared-memory multiprocessor (via fork), a set of workstations on a local area network (e.g., via `rsh`), or on an MPP (e.g., via POE commands on an IBM SP). Our MPICH-G implementation allows the same command syntax to be used even when starting programs across multiple MPPs of different architectures. We believe that it is a significant achievement that we can provide a similarly simple and uniform interface in much more complex grid environments.

3.2 Globus

Globus is a widely used toolkit for building wide area applications. The toolkit comprises a set of inter-related components, each providing services and associated APIs that address a distinct aspect of wide area computing [10]. Components developed to date are

1. the Nexus communication library, providing support for multimethod communication;
2. resource management services, providing uniform interfaces to local schedulers and support for brokering and co-allocation (see below);
3. security services, providing support for single sign-on, multiway security contexts, and interfaces to local security services;
4. file access services, providing staging services and uniform interfaces to files, regardless of location;
5. an Lightweight Directory Access Protocol (LDAP)-based information service, the Metacomputing Directory Service (MDS), providing uniform access to up-to-date information about Globus resource structure and state;
6. a fault detection service, providing a notification service for faulty processes; and
7. executable management services that support staging of executables to remote computers.

Globus has distinct local service, global service, and client components. At Globus sites, a small set of servers provide (deliberately simple) *local services* such as authentication, resource allocation, and status monitoring. In particular, a Globus Resource Allocation Manager (GRAM) implements a uniform interface to local resources (computers, networks, etc.) for authentication and allocation. Additional *global services*, defined in terms of these local services, provide more sophisticated functionality, such as resource brokering, co-allocation of resources, and fault detection. Finally, *client libraries* allow application programs and tools to invoke local and global services.

Globus toolkit components are designed to support the incremental development of grid-enabled tools and applications. In principle, the user should be able to take either an existing or new program and gradually make it more "grid-aware" by introducing additional services. Preliminary application experiences suggest that this incremental development methodology works well [10]. Various groups are using a similar methodology to apply Globus components in other tool projects (e.g., [1, 13]); however, MPICH-G is the most sophisticated such system constructed to date.

4 The MPICH-G Library

We briefly describe the techniques used to implement some of the MPICH-G capabilities listed in the introduction.

4.1 Startup: `mpirun` and the `machines` File

MPICH provides a standard command for starting MPI programs, namely, `mpirun`. This command specifies the number of processes that are to be created and can also provide flags relating to debugging and so forth.

On a parallel computer such as the IBM SP, the MPICH implementation of `mpirun` simply generates an appropriate job submission command to whatever scheduler is used to obtain access to the MPP. On the other hand, in a network of workstations environment, a `machines` file is accessed to determine which machines the MPI program should be started on. For example, the following file indicates that one process should be started on each of `donner` and `dalek`, and two processes on `pitcairn`.

```
donner
dalek
pitcairn 2
```


Our only change to the MPICH startup model is that we generalize the contents of the machines file to include resource manager (GRAM) names. For example, the following file names three such resource managers, at three different sites:

```
donner.mcs.anl.gov-fork 8
bonny.isi.edu-fork 8
moti4.ncsa.uiuc.edu-lsf 64
```

The MPICH-G implementation then uses the Globus information service, MDS, to perform a simple form of resource location, accessing MDS to determine detailed contact information (e.g., port numbers) for the specified resource managers. Hence, the user need not be concerned with low-level details regarding the physical location and interfaces of resources.

The user can build on this simple capability to implement more sophisticated resource location schemes. For example, rather than specifying node counts in the machines file, the user can perform an MDS search to determine how many nodes are available on each machine, and can rewrite the machines file appropriately. Or, the user can perform an MDS search to locate resource managers with particular properties (e.g., idle nodes and specified network bandwidth) and then place the names of those systems in the machines file.

4.2 Job Submission and Execution

Once the machines file has been read and resource manager contacts determined, the MPICH-G `mpirun` implementation calls a Globus-provided function called `globusrun` to manage the task of job submission and execution. This function uses a variety of Globus services and libraries, as follows:

Co-allocation. As noted above, the creation of a computation that spans multiple MPPs is a difficult problem. We must allocate resources on the selected computers, start processes, and link these processes into a computation. Different computers differ widely in the mechanisms used for resource allocation and process creation, so a first requirement is to negotiate the appropriate mechanisms at each site. A second concern is that startup can be a timeconsuming and error-prone activity; hence, we require techniques for detecting failure (e.g., via timeout) and synchronizing once startup completes. These two concerns are addressed via the use of the GRAM interface (discussed above) and an appropriate *co-allocator* library, respectively. MPICH-G uses the Dynamically-Updated Request Online Co-allocator (DUROC). DUROC submits requests, verifies correct startup, and provides functions that can

then be used to coordinate the various subjobs so as to create (in our current case) a single `MPI_COMM_WORLD` spanning all processes. The need to reserve resources at multiple sites simultaneously remains as a problem; which we are investigating in current work.

Authentication and authorization. A significant obstacle to the use of multiple distributed resources is that the user will typically have a distinct "trust relationship" (e.g., account), or even no prior trust relationship at all, at different sites. Hence, starting a program can be a frustrating process involving multiple logins. MPICH-G avoids this because the Globus Security Infrastructure supports single sign-on and automatic mapping (under site control) to appropriate local accounts. Public key technology is used to avoid the transfer of plaintext passwords.

Executable staging. Manual staging of executables is another painful activity. MPICH-G overcomes this obstacle by using the Globus "Global Access Secondary Storage" (GASS) service to stage executables to remote machines. Currently, this technique works only if the programmer has supplied an appropriate executable for each remote computer. In future work, the Globus group plans to investigate automated techniques for identifying and generating appropriate executables, for example by using compile servers.

Communication. As described in an earlier paper [8] which focused specifically on multimethod communication in MPICH-G, the Nexus communication library is used to provide access to multiple communication methods [9]: e.g., TCP/IP in the wide area, vendor-specific protocols within a computer, and shared memory within a cluster.

Monitoring, control, stdout. The `globusrun` utility used by `mpirun` also provides a number of other useful capabilities. Callbacks provided by GRAMs allow it to detect and report termination. Control functions provided by the GRAM API allow it to terminate a computation in the event of a user signal (control-C) or if a component fails. Finally, GASS mechanisms are used to collect standard output and error streams and route these back to the originating terminal.

5 Performance Studies

An empirical evaluation of a library such as MPICH-G should, ideally, address at least the following issues:

1. Startup costs: What is the cost of the authentication, authorization, resource location/allocation, and other management mechanisms? Are these mechanisms scalable?
2. Communication costs: What is the impact of the multimethod communication support on point-to-point and collective communication performance, for both simple benchmark programs and real applications, and in both homogeneous and heterogeneous environments?
3. Reliability: Are the management and communication mechanisms provided able to operate reliably in wide area environments?

We present here preliminary results for point-to-point communication performance in homogeneous systems; optimization in this configuration, and other measurements, are ongoing. We use the "ping-pong" benchmark programs provided with MPICH [15] to evaluate the performance of MPICH-G. We study performance on an IBM SP2 system at Lawrence Livermore National Laboratory (LLNL). This system runs AIX 4.3.1 and is configured with four-way SMP nodes with 332 MHz PowerPC 604e processors. This configuration provides 1.2 GB/s bandwidth to memory and 150 MB/s switch bandwidth. All communication measurements are between processors on different nodes.

We measured performance for five different communication libraries:

1. IBM-MPI, the nonthreaded IBM implementation of MPI.
2. IBM-MPL, the IBM implementation of MPL, the original communication library provided on the IBM SP.
3. MPICH-mpl, MPICH operating over the IBM MPL library.
4. Nexus, the Globus communication library (also operating over the IBM MPL library in this situation).
5. MPICH-G, MPICH-G operating over the Globus communication library (which in turn uses the IBM MPL library).

In addition, for each of these libraries we measured performance when operating over two different bindings for the IBM and IBM MPL library: one that uses the more efficient user space communication and one based on TCP/IP. Also, for Nexus and MPICH-G we evaluated the impact of two different values for the

"skip-poll" parameter, as discussed below. The results are presented in Tables 1 and 2.

In brief, we find that when using user space communication, MPICH-G incurs an overhead of 48 μ sec for a zero-length message (when skip poll=10K) and achieves 35 percent of the peak bandwidth achieved by IBM's MPI. These are certainly not good results, but nor are they dreadful, and on the basis of previous studies [12, 8], we believe that we understand the source of these overheads and know how to eliminate a significant part of them, by eliminating extra copies, improving memory management, and streamlining certain interfaces. Overall, we believe that we can achieve performance close to that of MPICH-mpl in most situations.

The user space results for Nexus and MPICH-mpl provide some insights into the nature of the overheads. The zero-byte latency for Nexus is 42 μ sec, while that for MPICH-mpl is only 32 μ sec; this difference reflects certain known overheads associated with the Nexus communication model and implementation [12]. But the bulk of the overhead (31 μ sec) is clearly associated with the layering of MPICH-G on Nexus, something that we have not optimized carefully. The bandwidth numbers for Nexus and MPICH-G are identical, indicating that the overheads here lie in Nexus. The source of this overhead is additional copies performed in the Nexus system on send and receive. These can be corrected, but the necessary optimizations have not yet been performed.

When using TCP/IP for communication, MPICH-G incurs a similar overhead for zero-length messages (69 μ sec) but now attains 61 percent of the bandwidth achieved by IBM's MPI. The overheads associated with the layering of MPICH-G over Nexus and the bandwidth behaviors seen for Nexus and MPICH-G are comparable to those seen in the user space case.

We comment finally on the significance of the skip poll parameter. As discussed elsewhere [9], the performance of multimethod systems that depend on polling to detect incoming communications can be sensitive to the frequency with which different interfaces are polled. In the current case, a user space poll is cheap (less than one μ sec), while an IP poll can cost 10s of microseconds. Hence, a simple round-robin strategy that polls the two interfaces in sequence will often delay the processing of incoming user space communications. We allow the user to control the polling strategy used by providing a parameter "skip-poll" that specifies how many "fast" polls are performed before a slow poll is performed. Hence, a very large skip-poll value such as 10,000 is a close approximation to the case when the slow protocol is not used at all, while skip-poll=0

Table 1. Preliminary performance results for MPICH-G: One-way message times on the LLNL IBM SP2

Communication Library	Skip poll	Latency (μ sec)	Time (μ sec) vs. Msg Size (bytes)					
			10	100	1K	10K	100K	1M
User space communication:								
IBM-MPI		25	27	32	64	284	1745	12714
IBM-MPL		24	26	30	63	235	1673	12681
MPICH-mpl		32	33	44	75	233	1630	12888
Nexus	10K	42	44	48	88	356	3944	35252
MPICH-G	10K	73	76	80	121	363	3249	35813
Nexus	0	161	162	167	224	701	6424	59886
MPICH-G	0	360	362	368	443	958	6458	57016
TCP/IP-based communication:								
IBM-MPI		131	134	143	251	976	4850	35272
IBM-MPL		129	133	141	251	718	4542	35061
MPICH-mpl		184	184	290	393	966	5800	35348
Nexus	10K	160	163	173	293	899	6993	57557
MPICH-G	10K	200	206	218	340	989	7058	58092
Nexus	0	287	289	294	430	1109	7856	62826
MPICH-G	0	530	544	558	693	1429	8141	62443

Table 2. Preliminary performance results for MPICH-G: Bandwidths on the LLNL IBM SP2

Communication Library	Skip poll	Latency (μ sec)	Bandwidth (KB/sec) vs. Msg Size (bytes)					
			10	100	1K	10K	100K	1M
User space communication:								
IBM-MPI		25	349	3034	15142	34381	55935	76809
IBM-MPL		24	370	3219	15396	41401	58358	77005
MPICH-mpl		32	292	2211	12975	41868	59882	75769
Nexus	10K	42	221	1995	10975	27366	24757	27701
MPICH-G	10K	73	128	1217	8067	26896	30051	27268
Nexus	0	161	60	583	4355	13918	15200	16312
MPICH-G	0	360	26	265	2201	10184	15121	17127
TCP/IP-based communication:								
IBM-MPI		131	72	681	3884	10003	20132	27686
IBM-MPL		129	73	688	3882	13594	21498	27853
MPICH-mpl		184	52	336	2481	10099	16834	27626
Nexus	10K	160	59	563	3331	10854	13964	16966
MPICH-G	10K	200	47	446	2864	9869	13835	16810
Nexus	0	287	33	331	2271	8801	12430	15543
MPICH-G	0	530	17	174	1407	6833	11994	15639

corresponds to round-robin polling. We see from Tables 1 and 2 that the round-robin strategy performs significantly worse than skip-poll=0. Fortunately, experience shows that even quite small skip-poll values can provide acceptable overheads while providing reasonable responsiveness for the different methods.

6 Future Work

We are working with colleagues to extend the MPICH-G implementation in a number of areas.

Shared-memory support. To date, we have explored the use of just two communication methods: user-space communications within an MPP and TCP/IP between MPPs. On computers such as the IBM SP, we can also exploit more efficient shared memory communications within SMP clusters, hence providing a total of three different communication methods. We are working with colleagues at USC/ISI to implement and evaluate this strategy.

Topology-aware communication operations. In heterogeneous grid environments, collective operations such as `MPI_REDUCE` can execute significantly faster if their implementation takes advantage of knowledge of the underlying system topology. For example, an `MPI_REDUCE` operation in the environment of Figure 1 might well first reduce within each SMP node, then within each MPP, and finally across MPPs. In order to implement such optimizations, the MPICH implementation requires information about the topology of the underlying machine. We are working with colleagues at LLNL to identify the required information and will extend the Globus device with additional functions that provide this information.

User-level communication structures can also take advantage of topology information. In principle, MPI's topology operations provide a basis for providing this information to applications. We plan to study whether these operations are indeed appropriate, or whether MPI extensions are needed to allow programmers to implement efficient applications in wide area environments.

Looking further into the future, we are interested in exploring more sophisticated techniques suitable for true wide area operation, for example exploiting Nexus support for multicast [21] and using network performance information (e.g., [22]) to adapt a combining tree structure in response to changing network loads.

MPI-2 extensions. The MPI-2 revisions to the MPI standard introduce a number of new features, including

single-sided operations, dynamic process creation and attachment, and parallel I/O. All three of these extensions can, in principle, be incorporated into MPICH-G easily. The Nexus communication library used in MPICH-G provides a single-sided communication operation as a primitive; Globus mechanisms support dynamic process creation and attachment; and a remote I/O binding for MPI-IO has already been developed. However, numerous details remain to be worked out in each of these areas, and the MPICH framework itself must be extended to support these new features.

7 Summary

We have described MPICH-G, an implementation of the Message Passing Interface that uses services provided by the Globus toolkit to allow the use of MPI in wide area environments. MPICH-G masks details of underlying networks and computer architectures so that diverse distributed resources can appear as a single "MPI_COMM_WORLD." Any arbitrary MPI application can be started on heterogeneous collections of machines simply by typing `mpirun`: authentication, authorization, executable staging, resource allocation, job creation; startup, and routing of stdout and stderr are all handled for free.

We believe that MPICH-G is interesting not only in its own right but also as a demonstration and test case for Globus services. MPICH-G was constructed by adapting MPICH, a widely used MPI implementation for workstations and MPPs. This adaptation involved the use of various Globus tools, for security, remote file access, synchronized startup, and multimethod communication. Relatively few changes to MPICH were required to support the use of these tools.

MPICH-G passes the MPICH test suite and is hence ready for broad distribution and use. Work is continuing on point-to-point performance optimization, application development, and research investigations relating to collective operation performance, network topology information, MPI-2 implementation, and other issues.

Acknowledgments

We gratefully acknowledge the contributions of Steven Tuecke, Brian Toonen, and Joe Bester to the design of the MPICH-G system; the meticulous work performed by Olle Larsson on MPICH-G performance evaluation; and the assistance of Bill Gropp and Rusty Lusk on MPICH implementation issues. We are also grateful to the members of the Globus project team

at Argonne National Laboratory and the University of Southern California's Information Sciences Institute for their help.

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; and by the National Science Foundation.

References

- [1] D. Abramson, R. Sasic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc. 4th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1995.
- [2] R. Butler and E. Lusk. Monitors, message, and clusters: The p4 parallel programming system. *Parallel Computing*, 20:547-564, April 1994.
- [3] Henri Casanova and Jack Dongarra. Netsolve: A network server for solving computational science problems. Technical Report CS-95-313, University of Tennessee, November 1995.
- [4] Jian Chen and Valerie Taylor. Mesh partitioning for distributed systems. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1998.
- [5] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The Network-Enabled Optimization System (NEOS) Server. Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, Illinois, 1996.
- [6] Tom DeFanti and Rick Stevens. Teleimmersion. In [11], pages 131-156.
- [7] G. Fagg, J. Dongarra, and A. Geist. PVMPI provides interoperability between MPI implementations. In *Proc. 8th SIAM Conf. on Parallel Processing*. SIAM, 1997.
- [8] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. A wide-area implementation of the Message Passing Interface. *Parallel Computing*, 1998. to appear.
- [9] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke. Managing multiple communication methods in high-performance networked computing systems. *Journal of Parallel and Distributed Computing*, 40:35-48, 1997.
- [10] I. Foster and C. Kesselman. The Globus project: A status report. In *Proceedings of the Heterogeneous Computing Workshop*, pages 4-18. IEEE Computer Society Press, 1998.
- [11] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
- [12] I. Foster, C. Kesselman, and S. Tuecke. The Nexus approach to integrating multithreading and communication. *Journal of Parallel and Distributed Computing*, 37:70-82, 1996.
- [13] D. Gannon, P. Beckman, E. Johnson, and T. Green. *Compilation Issues on Distributed Memory Systems*, chapter HPC++ and the HPC++Lib Toolkit. Springer Verlag, 1997.
- [14] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.
- [15] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22:789-828, 1996.
- [16] William Johnston. Realtime widely distributed instrumentation systems. In [11], pages 75-103.
- [17] Olle Larsson. Implementation and performance analysis of a high-order CEM algorithm in parallel and distributed environments. Master's thesis, University of Houston, 1998.
- [18] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proc. 8th Intl Conf. on Distributed Computing Systems*, pages 104-111, 1988.
- [19] Paul Messina. Distributed supercomputing applications. In [11], pages 55-73.
- [20] Reagan Moore, Chaitanya Baru, Richard Marciana, Arcot Rajasekar, and Michael Wan. Data-intensive computing. In [11], pages 105-129.
- [21] L. Moser, P. Melliar-Smith, D. Agarwal, R. Budhia, and C. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4), 1996.
- [22] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, Portland, Oregon, 1997. IEEE Press.

A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation

Ian Foster

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439, U.S.A.

Alain Roy

Department of Computer Science
The University of Chicago
Chicago, IL 60637, U.S.A.

Volker Sander

Central Institute for Applied Mathematics
Forschungszentrum Juelich GmbH
52425 Juelich, Germany

Abstract—Reservation and adaptation are two well-known and effective techniques for enhancing the end-to-end performance of network applications. However, both techniques also have limitations, particularly when dealing with high-bandwidth, dynamic flows: fixed-capability reservations tend to be wasteful of resources and hinder graceful degradation in the face of congestion, while adaptive techniques fail when congestion becomes excessive. We propose an approach to quality of service (QoS) that overcomes these difficulties by combining features of reservations and adaptation. In this approach, a combination of online control interfaces for resource management, a sensor permitting online monitoring, and decision procedures embedded in resources enable a rich variety of dynamic feedback interactions between applications and resources. We describe a QoS architecture, GARA, that has been extended to support these mechanisms, and use three examples of application-level adaptive strategies to show how this framework can permit applications to adapt both their resource requests and behavior in response to online sensor information.

I. INTRODUCTION

Network applications that need to achieve reliable end-to-end performance typically make use of either reservations or adaptation. When using reservations, applications usually specify quality of service (QoS) requirements when a connection is established and do not change them subsequently; the QoS system in turn guarantees that (modulo system failures or preemptions) the reservation will not be reduced during the lifetime of the application [1], [2]. In contrast, applications that use adaptation do not make reservations but instead adapt to the network conditions at hand by responding to some form of feedback, whether explicit (notification of network conditions) or implicit (noticing that bandwidth is low). Adaptation may occur when the application detects a problem or when the application is notified that a problem may exist [3], [4], [5], [6], [7].

Both reservations and adaptation have been proven effective in many situations, but also have significant limitations: particularly when dealing with high-end applications featuring high-bandwidth, dynamic flows. Fixed-capability reservations can waste bandwidth and do not permit graceful degradation in application performance when resource management policies mandate changes in allocations. Adaptive techniques inevitably fail when congestion reduces available resources below acceptable limits [8], [9].

In this paper, we describe an approach to QoS that combines

features of both reservations and adaptation to address the difficulties just noted. At the core of this approach is a QoS architecture in which resources are enhanced with:

- *online control* interfaces that allow applications, or agents acting on their behalf, to modify resource characteristics (e.g., reservations) dynamically;
- *sensors* that allow applications (and agents) to detect when adaptation is required; and
- *decision procedures* that support the expression of a rich set of resource management policies.

These mechanisms in turn enable a wider range of application-level adaptation strategies than are supported in other architectures. For example, online control of reservations allows applications to request premium service when adaptive techniques fail to deliver; monitoring of reservations that change as a result of decision procedures embedded in resource managers allows for graceful degradation in application performance in response to preemption.

To explore these ideas, we have incorporated such mechanisms into a QoS architecture developed in previous work—the Globus Architecture for Reservation and Allocation (GARA) [10], [11]. We have completed a prototype implementation of this enhanced architecture, which has been deployed by ourselves and others on local and national testbeds.

We hypothesize that the mechanisms and associated control and information flows provided by this extended GARA architecture can be exploited to obtain more efficient resource usage than in purely reservation-based or application-based approaches, as applications can vary reservations and rates; to provide more flexible resource allocation strategies, as resources can change allocations over the course of a reservation; and to deliver more robust application performance, as applications can detect and respond to changes in allocations and resource state.

As a first step towards testing this hypothesis, we have used GARA mechanisms to implement three different adaptive strategies. The first two use a flow-specific packet loss sensor to adapt bandwidth requests to the QoS system in order to meet performance targets, for UDP and TCP flows, respectively; the third uses a sensor that provides information on changes in reserva-

tion level (as a result of preemption) to adapt transmission rate for bulk data transfer applications. In each case, we present novel decision procedures and demonstrate that we can deliver interesting adaptive behaviors via a combination of online monitoring and control.

In the rest of this paper, we review the QoS requirements of high-end applications, describe our enhanced GARA architecture, and present our three adaptive strategies and the experimental studies that we have performed to evaluate their effectiveness. We conclude with a brief discussion of related and future work.

II. MOTIVATION: HIGH-END APPLICATIONS

We are interested in providing QoS mechanisms for *high-end network applications* [11], in which individual flows can have high bandwidth, from a few megabits per second (Mb/s) to many tens or hundreds of Mb/s; there may be complex mixes of flows, from low bandwidth to high bandwidth and from low latency to high latency; and flows may change their requirements dynamically throughout their lifetime.

Applications with these characteristics arise in such areas as distance visualization, analysis of petabyte-scale scientific databases, online control of scientific instrumentation, and teleimmersion [12]. For illustrative purposes, we examine a teleimmersion example in more detail. Consider two or more users at geographically separate locations who are exploring collaboratively a three-dimensional visualization of experimental data. As in other telecollaboration systems, we have a number of streams with fairly constant rate and low to moderate bandwidth: audio and video streams for communication, and jitter- and latency-sensitive streams for the tracking data indicating user movements in the virtual space. In addition, we have streams with higher bandwidth and often variable rates, used for visualization data and (in some cases) database updates. Visualization data is calculated from the data set, and a representation of it, perhaps a set of polygons for rendering, is transmitted [13]. The actual amount of data sent depends on both the data being visualized and user actions, which may include zooming and movement in space and time. Contention for shared resources such as disk and CPU can also affect the transmission rate.

These characteristics place substantial demands on both network infrastructure and applications. For example, consider a situation in which several teleimmersion sessions are in operation simultaneously, while other groups are concurrently attempting to perform high-speed bulk-data transfers over the same network infrastructure, perhaps to stage data required for an experiment later in the day. With today's protocols and services, no group would obtain acceptable service.

We believe that concerns such as these require that resource providers be able to specify and implement flexible resource allocation policies. For example, in the situation just noted, resource providers might allocate resources to different teleimmersion sessions and bulk-data transfers differentially. Teleimmersion session *A* might have priority, while sessions *B* and *C* would be guaranteed some minimum service. Bulk-data transfers *D* and *E* would have lowest instantaneous priority but would

be guaranteed service in terms of another "terabytes per hour" metric.

We also believe that a policy-driven framework of this sort can be effective only if applications themselves are provided with the information and control flows required to detect and adapt to policy-driven changes in resource allocations. For example, a teleimmersion session could respond to reduced (increased) resource availability by reducing (increasing) video rates or introducing (eliminating) data compression to noncritical users, while a bulk-data transfer could reduce (increase) its sending rate. The architecture that we present in this paper enables these sorts of adaptation.

III. RESERVATION AND ADAPTATION COMBINED

Effective adaptive control requires three distinct mechanisms.

In the language of [14], these are

- *actuators* that permit online control, for example, of resource allocations or application behavior;
- *sensors* that permit monitoring, for example, of resource allocations or application behavior; and
- *decision procedures* that allow entities to respond to sensor information, by invoking actuators.

As illustrated in Figure 1, these three elements act in concert to achieve adaptive control. For example, a sensor might signal a nonzero loss rate associated with a flow at a router. A decision procedure in the associated application can then execute to determine whether to reduce the sending rate or, alternatively, generate a request to a resource manager to create (or increase) a reservation for that flow, hence invoking an actuator.

In this section, we first provide an overview of the GARA architecture and then explain how we have extended it to support these three mechanisms.

A. GARA Overview

The Globus Architecture for Reservation and Allocation provides advance reservations and end-to-end management for quality of service on different types of resources, including networks, CPUs, and disks [10], [11].

A GARA system comprises a number of *resource managers* that each implement reservation, control, and monitoring operations for a specific resource. Resource managers can and have been implemented for a variety of resource types, hence the use of the term "resource manager" rather than the more specific "bandwidth broker" favored in the networking literature [15]. Uniform interfaces allow applications to express QoS needs for different types of resources in similar ways, hence simplifying the development of end-to-end QoS management strategies. Mechanisms provided by the Globus toolkit are used for secure authentication and authorization of all requests to resource managers. An information service allows applications to discover resource properties such as current and future availability.

The work described in this article involves just a single type of resource manager, namely, one that uses differentiated services mechanisms [16] to implement network QoS. This resource manager uses the *expedited forwarding* per-hop behavior (PHB), as specified by the Internet Engineering Task Force's

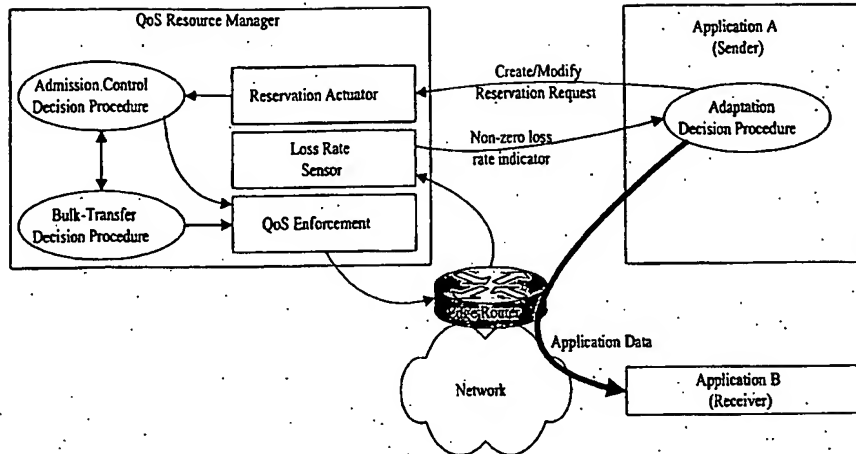


Fig. 1. An example of how actuators, sensors, and decision procedures may be combined to provide adaptive control. We illustrate reservation adaptation in Application A, occurring as a result of a packet loss notification received from a router via the resource manager. The operation of this strategy is described in the text, as is the bulk transfer decision procedure that is also shown.

(IETF) Working Group on Differentiated Services, to provide a premium service. With careful admission control at the edge of the network, it is possible to build a network QoS system with reasonably strong bandwidth guarantees, even though traffic is treated as an aggregate in the core of the network.

The resource manager enables reservation requests (see below) by configuring the routers that it controls. In particular, it configures the ingress routers to classify, police, mark, and potentially shape, all packets that belong to a flow for which a reservation has been authorized, as is normally done for differentiated services. The expedited forwarding per-hop behavior drops packets that exceed the reservation, but allows small bursts of excess traffic using a token-bucket mechanism.

B. Actuators: Online Control

A first prerequisite for adaptation is support for online control of resource characteristics. (We are also interested in online control of application behavior, but that topic is beyond the scope of this article.) GARA supports this requirement directly via control functions that allow an application—or an agent acting on its behalf—to make and subsequently modify QoS reservations.

In the case of the network resources considered in this article, an application request to the resource manager specifies a start time and duration for the desired reservation; the IP addresses of the end hosts that will be communicating; the bandwidth required for the reservation; and the network protocol that will be used (TCP or UDP) [10]. Since reservations may be made in advance, not all information may be known at the time the reservation is made. In particular, an application may not know what port numbers will be used for communication until network communications begin. Therefore, GARA provides a “bind” operation, which simultaneously “claims” the reservation and provides this run-time information.

Both immediate and advance reservations are supported. Advance reservations simplify co-scheduling of scarce resources

and help to ensure that resources are available for important events, such as scientific experiments.

GARA allows third parties to make, monitor, and modify reservations on behalf of an application. This capability allows us to separate adaptation logic from an application proper; in the case of advance reservations, it means that an application need not be running when a reservation is made. For brevity, we frame subsequent discussion as if only applications manipulate reservations; however, in practice, a third party can always be substituted.

C. Sensors

A second requirement for adaptive control is that we be able to determine the state of system components and detect state changes. This capability is provided via sensors associated with system entities to which other entities can subscribe, with notifications provided via some form of event service or callback mechanism. We have implemented two such sensors in our GARA prototype.

C.1 Loss rate sensor.

This sensor provides applications with information on packet loss rate in the network. This information can serve to indicate the application is either sending too fast or has an inadequate reservation.

We measure packet loss rates at the first hop router: that is, the router at which initial policing is performed by our differentiated services implementation. Our resource manager periodically queries this router, which because of its classification and policing role is able to provide statistics about the number of packets that have exceeded a flow’s reservation.

The query to the router returns the number of packets that conformed to the reservation and were not dropped (p_c) and the number of packets that exceeded the reservation and were dropped (p_e), both of these quantities being since the last time

the statistics were queried. If the resource manager detects a nonzero p_e value then it generates a callback to notify any subscribed processes that packet loss has occurred. This callback specifies both an estimated loss percentage and the currently unallocated bandwidth; an application might use the latter quantity as a guide when deciding whether to respond to a packet loss notification by attempting to increase its reservation vs. changing its behavior.

In computing the estimated bandwidth, we must deal with the complicating factor that the router uses a token bucket of size p_b to allow small bursts. The router updates its statistics only periodically (roughly every 10 seconds) and the resource manager cannot know if the token bucket was full or empty when the statistics were gathered. To avoid persistent underestimates of loss rates, we assume that the token bucket is at least half-full and reduce the number of conforming packets correspondingly. This adjustment is reflected in our formula for estimated fraction of packets that were dropped:

$$P = \frac{p_e}{p_c + p_b/2 + p_e}$$

We describe in Section IV how this sensor can be used to modify QoS reservations to meet application requirements, for both UDP and TCP flows.

C.2 Reservation change sensor.

Our second sensor is used to publish information about changes in resource allocations. The reason for these changes is described in the next subsection; here we note simply that we have a sensor capable of communicating such changes to interested entities.

D. Decision Procedures

The third component of an adaptive control architecture comprises the decision procedures that invoke actuators in response to sensor data.

In our environment, such decision procedures can occur in multiple locations. They clearly arise in applications, and indeed we give three such examples below. Decision procedures can also occur in resource managers; this can lead to interesting interactions.

Decision procedures may be invoked within a GARA resource manager at a number of points. Following authentication, an incoming request is first authorized and then executed. Decision procedures may be invoked at both stages: for example, to determine whether a request should be granted, in the first instance, and to reallocate resources in the second instance if the newly authorized reservation oversubscribes available resources.

To explore these ideas and demonstrate our ability to incorporate decision procedures in resource managers, we have implemented the following simple but highly effective procedure.

D.1 Bulk-data transfer procedure.

As noted above, bulk-data transfer (BDT) operations have service requirements expressible in terms of "terabytes per hour"

rather than "Mb/s." Satisfying such requirements in the face of congestion can require the use of premium service but need not always pre-empt other applications requiring premium service.

Our BDT decision procedure is designed to exploit this observation. In effect, it implements two classes of premium service, foreground and background, within a single premium service class. It does this by applying the following simple decision rules when processing requests to create, bind, or terminate reservations.

1. *Create foreground reservation:* Creation of a foreground reservation is authorized if at no time during the reservation period the sum of all foreground reservations would exceed the total available premium bandwidth.
2. *Bind foreground reservation:* Binding of a foreground reservation results in the requested bandwidth being allocated to the appropriate flow. If necessary, premium bandwidth is preempted from background flow(s), with callbacks being generated to notify interested parties.
3. *Cancel reservation:* The freed bandwidth is allocated to background flows with inadequate allocations, if any such exist, and callbacks are generated.
4. *Create background reservation:* Creation of a background reservation is always allowed.
5. *Bind background reservation:* Binding of a reservation results in a "fair share" of the unallocated premium bandwidth being allocated to the appropriate flow. (See below for a description of how this fair share is calculated.)

We describe below how an application can use the reservation change sensor triggered by this decision procedure to achieve sustained BDT rates without impeding foreground flows.

IV. APPLICATION-LEVEL ADAPTATION PROCEDURES

We now describe the three application-level adaptation procedures that we have developed to date.

A. The GARA Testbed

All experiments reported below were performed in the testbed shown in Figure 2. The testbed consists of three Cisco 7507 routers interconnected with 155 Mb/s (OC-3) ATM. Hosts are connected to the routers with 100 Mb/s switched Ethernet. All hosts used in our tests were Sun Ultra 60s. In addition, virtual circuits to several remote sites permit wide area experiments.

Cisco's Modular QoS command line interface (MQC) is used for two different purposes. On the ingress interfaces to the network, it is used to classify, police, and mark packets. Within the interior of the network, it is used to enable Weighted Fair Queuing (WFQ) to give priority to marked packets.

B. Adaptive QoS Reservations: UDP Flows

We first describe how adaptive techniques can be used to determine the bandwidth reservation required to support a particular UDP flow. The motivation for this use of adaptation is that many application developers have no knowledge or QoS mechanisms or of the principles by which QoS parameters are determined. We show that information provided by a simple packet loss rate sensor can be used to guide a decision procedure that

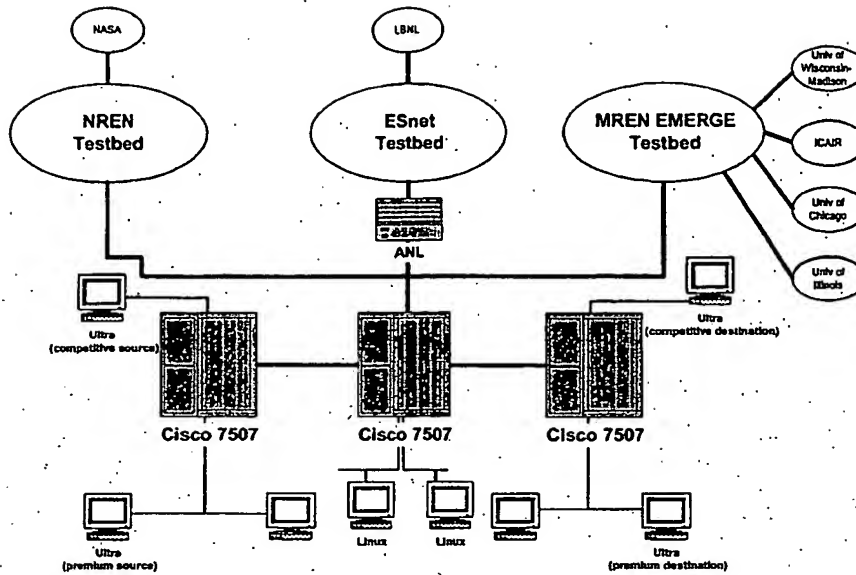


Fig. 2. The GARA Network Testbed (GARNET). The core of the testbed consists of three Cisco 7507 routers. There are several computers on each end of the testbed, more than shown here. Note the connections to three wide area networks.

sets bandwidth reservations adaptively, increasing reservations until loss rates reach zero. This decision procedure can be incorporated in an application or in a separate agent.

Our decision procedure uses information provided by the packet loss rate sensor described in Section III-C. Recall that this sensor periodically generates an estimate of the fraction of packets dropped, P ; hence, $1 - P$ is the fraction of packets that conformed to the reservation. Our decision procedure calculates what reservation would have been needed to make such that no packets would have been dropped, as follows:

$$R_n(1 - P) = R_o$$

or

$$R_n = \frac{R_o}{1 - P}$$

where R_o is the old reservation and R_n is the new reservation.

To evaluate the effectiveness of this strategy, we performed experiments as follows. In order to obtain a replicable experiment, we used as our application a test program that sends UDP traffic at a user-specified rate across our testbed.

Results for two similar experiments are superimposed in Figure 3. In each case, the application made an initial reservation for 2500 kilobytes per second (KB/s) but then sent data at a higher rate: in the first case at 4000 KB/s and in the second case at 8000 KB/s. As described before, the first router classified, policed, and marked traffic. Because the router allows small bursts, the application initially was able to send slightly faster than the reservation allowed, but then the data rate settled down to a constant 2500 KB/s.

Our loss rate sensor is implemented by the GARA resource manager, which queries the router every ten seconds and provides feedback to the application for every query except the first. (The first query is not reported to the application because we wish to gather statistically sufficient data.) As the resource manager and application are not synchronized in any way, we should not be surprised that the feedback arrives at slightly different times in the two cases: at 16 seconds and 22 seconds, respectively.

It is clear from Figure 3 that the UDP application was able to adapt quickly in these experiments. However, the poor temporal resolution offered by our routers means that adaptation need not always work so well. For example, if the router statistics were gathered just as a series of packets were starting to be dropped, an unrepresentative result may be reported to the application. However, this problem would be compensated for after another round of adaptation. In addition, our router updates statistics only every ten seconds, which limits the frequency at which the resource manager can check them.

C. Adaptive QoS Reservations: TCP Flows

We should not be surprised that it is possible to determine UDP transmission rates by monitoring packet loss information, given that UDP does not perform congestion control. Implementing a comparable adaptive strategy for TCP is significantly more complex because of TCP's self-clocking mechanisms. Data that an application attempts to write into a socket buffer with a specific rate may not be transported immediately because TCP's sliding window protocol requires that acknowledgments be received before further data is sent. Also, TCP slows its sending rate when it believes it has encountered congestion. (In our case, TCP has not encountered congestion, but

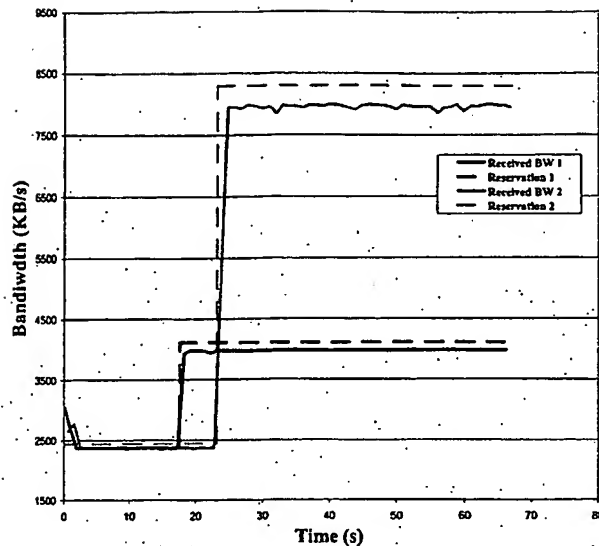


Fig. 3. Performance of our UDP reservation adaptation strategy in two different cases. In the first case, the application is sending at 4000 KB/s while in the second it is sending at 8000 KB/s. In both cases, an initial reservation of approximately 2500 KB/s is corrected after a single round of adaptation.

an aggressive QoS policing mechanism.) Nevertheless, TCP is used extensively in the applications that interest us, and so it is important to support TCP if we can.

Because of these difficulties, our decision procedure for TCP does not attempt to derive the transmission rate from the packet loss rate ratio. Instead, it uses a search procedure to determine the correct rate. When the packet loss rate sensor signals that packets have been dropped, we simply double the reservation. Once the reservation is large enough, we perform a binary search between the current reservation and the previous reservation until we arrive at a reservation that works and that has not changed from the previous reservation by more than five percent.

Figure 4 illustrates the results that we obtain with this heuristic. We see that the search takes some time to adapt but eventually comes close to a correct value. The time delay is largely because statistics on dropped packets are reported only every 10 seconds on our routers. Clearly, decreasing that interval would improve the adaptation time. Nevertheless, even this relatively long adaptation time is quite acceptable for many of our long-lived target applications.

There are a couple of possibilities for improving upon this search. One possibility is to change the initial doubling by estimating the correct multiplier from the percentage of dropped packets, much as we did in the UDP case. We have performed extensive experiments with such techniques but have not yet succeeded in identifying a good estimate for the multiplier, because of TCP's complex behavior. Recent work proposes modifying TCP's windowing algorithm to be aware of reservation rates [17], [18].

It may be possible to adapt more quickly by monitoring closer to the application. In particular, if the application used an instru-

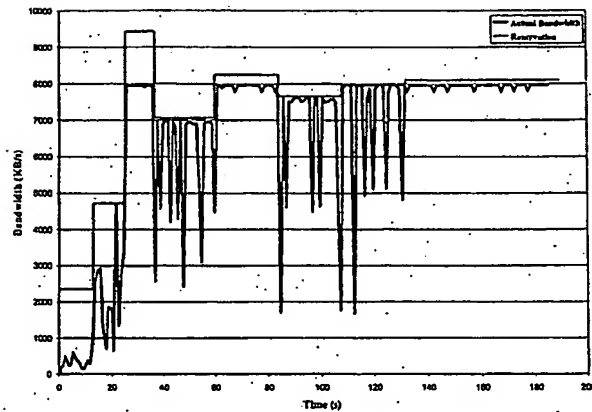


Fig. 4. An example of how our search-based reservation rate strategy determines the correct reservation to use for a TCP application. The application doubled its reservation request twice, then narrowed its reservation with a binary search. The difference between the gross and net reservations is due to packet overheads.

mented TCP library that could measure the rate at which the application was attempting to send, it could use the adjustment to adapt much more quickly. However, this strategy requires modifications to the application: our heuristics have the advantage of being usable by a third party agent.

D. A Bulk-Data Transfer Application

Our third example of an application-level adaptation procedure uses our BDT reservation-change sensor to guide rate adaptation for BDT applications. As described in Section III-D, this sensor signals changes in background flow reservations due to preemption by (or termination of) higher-priority foreground flows. Our decision procedure simply adapts the transmission rate of the TCP-based bulk data transfer application in order to achieve throughput close to the bandwidth allocated to the BDT flow. Note that in the absence of this decision procedure, the achieved throughput would tend to be extremely low, because preemption lowers the background flow's reservation and packets that exceed the reservation are dropped, therefore triggering TCP's backoff algorithms.

For our experiments, this decision procedure was incorporated into a QoS-aware TCP-based BDT application. Figure 5 shows results obtained in a wide area testbed between Argonne National Laboratory and Lawrence Berkeley National Laboratory. At about time 5, the (background) BDT application began and was assigned all of the premium bandwidth, 25 MB/s. At approximately times 40 and 100, a foreground reservation began and the BDT reservation was reduced. When the foreground reservations ended, the background reservation was increased. Notice that at time 15, competitive UDP traffic began but does not interfere with either the foreground or background reservations.

These results show that we are successful in adapting the BDT flow in response to information concerning preemption by foreground flows. Apart from a few artifacts, the BDT flow main-

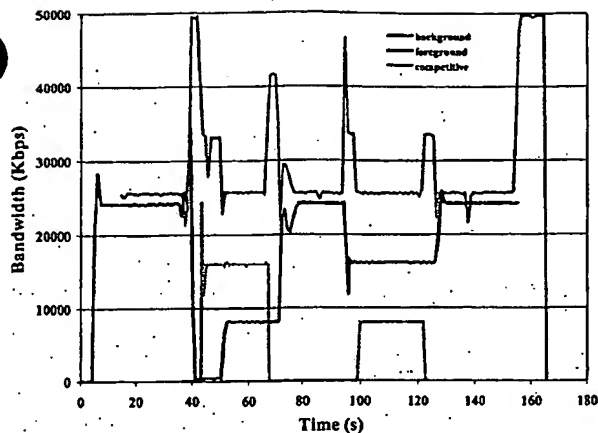


Fig. 5. An example of bulk-transfer in our wide area testbed. See text for details.

tains data transfer at a rate close to the amount of premium bandwidth allocated to that flow. The artifacts can be explained as follows. First, we see that each time the BDT reservation is reduced, the BDT rate drops momentarily more than expected and then recovers. We attribute this behavior to the fact that TCP shrinks its window size when packets are dropped (when the reservation is changed before the application adapts), either by falling into its slow-start phase or into its congestion avoidance phase.

In addition, the application is using large socket buffers to obtain high performance over the wide area testbed and when it enters slow-start mode (because packets have been dropped once the reservation decreases) these socket buffers quickly fill up. As TCP increases its congestion window size exponentially during the slow-start phase, data is immediately available to send; and TCP sends the data as increasingly larger bursts, until the socket buffer is emptied. Because the former congestion window size did not reflect the actual amount of data transmitted, the length of the slow-start phase after a drop is too long, therefore, data is initially sent too rapidly for the updated router configuration, forcing packets to be dropped and TCP to go into slow-start mode again, until the congestion window becomes more appropriate. This effect is magnified by the larger bandwidth-delay product and hence larger socket buffers (1 MB in this case) in the wide area network.

V. RELATED WORK

There has been a great deal of research on rate adaptation for network applications when reservation mechanisms are not present. For example, Goel et al. [7] describe a modular framework that provides feedback for not only network streams but also CPU scheduling. The present paper takes its terminology of actuators, sensors, and decision procedures from another feedback infrastructure, Autopilot [14], which has been used for dynamic performance tuning in various settings, including I/O. Our approach follows the concept of detaching the "controller" from the application, as proposed in [3].

Implementing QoS-aware middleware is addressed in several

projects. The Adaptive Quality of Service Architecture for distributed multimedia applications (AQUA) [19] introduces abstract interfaces for QoS measurements and negotiation. However, this work focuses on ATM-connections and how to ensure QoS under competition on the end-system.

The Quartz architecture [20] provides a CORBA-based QoS framework. It introduces agent-based adaption and a resource trader, called a balancing agent, which tries to compensate for the loss of resources by increasing the amount requested.

VI. CONCLUSIONS AND FUTURE WORK

We have argued that advanced network applications such as teleimmersion, bulk data transfer, and distance visualization can benefit from mechanisms that enable the coordinated use of reservation and adaptation, via support for dynamic feedback among entities involved in making resource management decisions. We have described an implementation of such mechanisms within the GARA resource management architecture. In this implementation, sensors associated with resource and resource managers permit application-level monitoring of resource state and reservation status, while online control mechanisms enable adaptive control of reservations. We have used these mechanisms to develop three different application-level adaptive control mechanisms: two that use loss rate information to adapt reservations and one that uses reservation state information to adapt transmission rate.

We find these initial results encouraging, but recognize that much more work remains to be done. For example, we would like to experiment with more sophisticated resource-side allocation policies and determine to what extent applications can adapt to these policies in interesting ways. In more complex multidomain environments, performance feedback and adaptation become more complex, not least because relevant sensor information may not be easily accessible. Finally, experimentation with a wider range of applications is required.

ACKNOWLEDGMENTS

We gratefully acknowledge assistance given by Linda Winkler and Becca Nitzen with the testbed used in these experiments and by Andy Adamson who wrote the UDP traffic generator. Numerous discussions with our colleagues Gary Hoo, Bill Johnston, Carl Kesselman, and Steven Tuecke have helped shape our approach to quality of service. We also thank Cisco Systems for an equipment donation that allowed the creation of the GARNET testbed. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the National Science Foundation; and by the NASA Information Power Grid program.

REFERENCES

- [1] L. Wolf and R. Steinmetz, "Concepts for reservation in advance," *Kluwer Journal on Multimedia Tools and Applications*, vol. 4, May 1997.
- [2] D. Ferrari, A. Gupta, and G. Ventre, "Distributed advance reservation of real-time connections," *ACM/Springer Verlag Journal on Multimedia Systems*, vol. 5, no. 3, 1997.
- [3] B. Li and K. Nahrstedt, "A Control-based Middleware Framework for Quality of Service Adaptations," *IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms*, June 1999.
- [4] B. Li and K. Nahrstedt, "QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications," in *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000)*, 2000.
- [5] X. Wang and H. Schulzrinne, "Comparison of Adaptive Internet Multimedia Applications," *Institute of Electronics, Information and Communication Engineers Transactions*, vol. E82-B, pp. 806-818, June 1999.
- [6] D. Sisalem and H. Schulzrinne, "The Loss-Delay Adjustment Algorithm: A TCP-friendly Adaptation Scheme," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, July 1998.
- [7] A. Goel, D. Steere, C. Pu, and J. Walpole, "Adaptive Resource Management Via Modular Feedback Control," Tech. Rep. 99-03, Oregon Graduate Institute, Computer Science and Engineering, Jan. 1999.
- [8] W. Almesberger, J. L. Boudec, and T. Ferrari, "Scalable Resource Reservation for the Internet," in *IEEE Conference on Protocols for Multimedia Systems -Multimedia Networking*, Nov. 1997.
- [9] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," in *18th IEEE Real-Time System Symposium*, 1997.
- [10] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture That Supports Advance Reservations and Co-Allocation," in *International Workshop on Quality of Service*, pp. 27-36, June 1999.
- [11] I. Foster, A. Roy, V. Sander, and L. Winkler, "End-to-End Quality of Service for High-End Applications," tech. rep., Argonne National Laboratory, 1999. <http://www.mcs.anl.gov/qos/qos.papers.htm>.
- [12] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
- [13] I. Foster, J. Insley, G. von Laszewski, C. Kesselman, and M. Thiebaut, "Distance Visualization: Data Exploration on the Grid," *IEEE Computer Magazine*, pp. 36-43, Dec. 1999.
- [14] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, "Autopilot: Adaptive Control of Distributed Applications," in *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, IEEE Computer Society Press, 1998.
- [15] K. Nichols, V. Jacobson, and L. Zhang, "A Two-Bit Differentiated Services Architecture for the Internet," *Internet RFC 2638*, July 1999.
- [16] S. Blake, D. Black, M. Carlson, M. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *Internet RFC 2475*, 1998.
- [17] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Understanding and Improving TCP Performance Over Networks with Minimum Rate Guarantees," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 173-187, Apr. 1999.
- [18] I. Yeom and A. N. Reddy, "Realizing Throughput Guarantees in a Differentiated Services Network," in *IEEE Int. Conf. on Multimedia Computing and Systems*, pp. 372-376, June 1999.
- [19] K. Lakshman and R. Yavatkar, "Integrated CPU and Network I/O QoS Management in an End-System," *Intel Architecture Labs and University of Kentucky in Computer Communications Journal, Special Issue on Quality of Service in Distributed Systems*, vol. 21, Apr. 1997.
- [20] F. Siqueira and V. Cahill, "Delivering QoS in Open Distributed Systems," in *Proceedings of the 7th IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS'99)*, Dec. 1999.

The Physiology of the Grid

An Open Grid Services Architecture for Distributed Systems Integration

Ian Foster^{1,2} Carl Kesselman³ Jeffrey M. Nick⁴ Steven Tuecke¹

¹ Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439

² Department of Computer Science, University of Chicago, Chicago, IL 60637

³ Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292

⁴ IBM Corporation, Poughkeepsie, NY 12601

foster@mcs.anl.gov carl@isi.edu jnick@us.ibm.com tuecke@mcs.anl.gov

Abstract

In both e-business and e-science, we often need to integrate services across distributed, heterogeneous, dynamic “virtual organizations” formed from the disparate resources within a single enterprise and/or from external resource sharing and service provider relationships. This integration can be technically challenging because of the need to achieve various qualities of service when running on top of different native platforms. We present an *Open Grid Services Architecture* that addresses these challenges. Building on concepts and technologies from the Grid and Web services communities, this architecture defines a uniform exposed service semantics (the *Grid service*); defines standard mechanisms for creating, naming, and discovering transient Grid service instances; provides location transparency and multiple protocol bindings for service instances; and supports integration with underlying native platform facilities. The Open Grid Services Architecture also defines, in terms of Web Services Description Language (WSDL) interfaces and associated conventions, mechanisms required for creating and composing sophisticated distributed systems, including lifetime management, change management, and notification. Service bindings can support reliable invocation, authentication, authorization, and delegation, if required. Our presentation complements an earlier foundational article, “The Anatomy of the Grid,” by describing how Grid mechanisms can implement a service-oriented architecture, explaining how Grid functionality can be incorporated into a Web services framework, and illustrating how our architecture can be applied within commercial computing as a basis for distributed system integration—within and across organizational domains.

This is a DRAFT document and continues to be revised. The latest version can be found at <http://www.globus.org/research/papers/ogsa.pdf>. Please send comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

Table of Contents

1	Introduction.....	3
2	The Need for Grid Technologies	5
2.1	The Evolution of Enterprise Computing.....	5
2.2	Service Providers and Business-to-Business Computing	6
3	Background.....	7
3.1	The Globus Toolkit.....	7
3.2	Web Services	8
4	An Open Grid Services Architecture	10
4.1	Service Orientation and Virtualization	10
4.2	Service Semantics: The Grid Service	12
4.2.1	Upgradeability Conventions and Transport Protocols	12
4.2.2	Standard Interfaces.....	13
4.3	The Role of Hosting Environments	14
4.4	Using OGSA Mechanisms to Build VO Structures	15
5	Application Example	17
6	Technical Details.....	18
6.1	The OGSA Service Model.....	18
6.2	Creating Transient Services: Factories	20
6.3	Service Lifetime Management.....	20
6.4	Managing Handles and References.....	21
6.5	Service Information and Service Discovery	22
6.6	Notification	23
6.7	Change Management	24
6.8	Other Interfaces.....	24
7	Network Protocol Bindings.....	24
8	Higher-Level Services	25
9	Related Work	25
10	Summary	26
	Acknowledgments.....	27
	Bibliography	27

1 Introduction

Until recently, application developers could often assume a target environment that was (to a useful extent) homogeneous, reliable, secure, and centrally managed. Increasingly, however, computing is concerned with collaboration, data sharing, and other new modes of interaction that involve distributed resources. The result is an increased focus on the interconnection of systems both within and across enterprises, whether in the form of intelligent networks, switching devices, caching services, appliance servers, storage systems, or storage area network management systems. In addition, companies are realizing that they can achieve significant cost savings by outsourcing nonessential elements of their IT environment to various forms of service providers.

These evolutionary pressures generate new requirements for distributed application development and deployment. Today, applications and middleware are typically developed for a specific platform (e.g., Windows NT, a flavor of Unix, a mainframe, J2EE, Microsoft .NET) that provides a hosting environment for running applications. The capabilities provided by such platforms may range from integrated resource management functions to database integration, clustering services, security, workload management, and problem determination—with different implementations, semantic behaviors, and APIs for these functions on different platforms. But in spite of this diversity, the continuing decentralization and distribution of software, hardware, and human resources make it essential that we achieve desired qualities of service (QoS)—whether measured in terms of common security semantics, distributed workflow and resource management, performance, coordinated fail-over, problem determination services, or other metrics—on resources assembled dynamically from enterprise systems, service provider systems, and customer systems. We require new abstractions and concepts that allow applications to access and share resources and services across distributed, wide area networks.

Such problems have been for some time a central concern of the developers of distributed systems for large-scale scientific research. Work within this community has led to the development of *Grid technologies* [30, 34], which address precisely these problems and which are seeing widespread and successful adoption for scientific and technical computing.

In an earlier article, we defined Grid technologies and infrastructures as supporting the sharing and coordinated use of diverse resources in dynamic, distributed “virtual organizations” (VOs) [34]. We defined essential properties of Grids and introduced key requirements for protocols and services, distinguishing among *connectivity* protocols concerned with communication and authentication, *resource* protocols concerned with negotiating access to individual resources, and *collective* protocols and services concerned with the coordinated use of multiple resources. We also described the Globus Toolkit¹ [29], an open source reference implementation of key Grid protocols that supports a wide variety of major e-science projects.

Here we extend this argument in three respects to define more precisely how a Grid functions and how Grid technologies can be implemented and applied. First, while [34] was structured in terms of the protocols required for interoperability among VO components, we focus here on the nature of the *services* that respond to protocol messages. We view a Grid as an extensible set of *Grid services* that may be aggregated in various ways to meet the needs of VOs, which themselves can be defined in part by the services that they operate and share. We then define the behaviors that such Grid services should possess in order to support distributed systems integration. By stressing functionality (i.e., “physiology”), this view of Grids complements the previous protocol-oriented (“anatomical”) description.

¹ Globus Project and Globus Toolkit are trademarks of the University of Chicago.

Second, we explain how Grid technologies can be aligned with Web services technologies [40, 47] to capitalize on desirable Web services properties, such as service description and discovery; automatic generation of client and server code from service descriptions; binding of service descriptions to interoperable network protocols; compatibility with emerging higher-level open standards, services and tools; and broad commercial support. We call this alignment—and augmentation—of Grid and Web services technologies an *Open Grid Services Architecture* (OGSA), with the term *architecture* denoting here a well-defined set of basic interfaces from which can be constructed interesting systems, and *open* being used to communicate extensibility, vendor neutrality, and commitment to a community standardization process. This architecture uses the Web Services Description Language (WSDL) to achieve self-describing, discoverable services and interoperable protocols, with extensions to support multiple coordinated interfaces and change management. OGSA leverages experience gained with the Globus Toolkit to define conventions and WSDL interfaces for a *Grid service*, a (potentially transient) stateful service instance supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization. OGSA also defines interfaces for the discovery of Grid service instances and for the creation of transient Grid service instances. The result is a standards-based distributed service system (we avoid the term distributed object system due to its overloaded meaning) that supports the creation of the sophisticated distributed services required in modern enterprise and interorganizational computing environments.

Third, we focus our discussion on commercial applications rather than the scientific and technical applications emphasized in [30, 34]. We believe that the same principles and mechanisms apply in both environments. However, in commercial settings we need, in particular, seamless integration with existing resources and applications, and with tools for workload, resource, security, network QoS, and availability management. OGSA's support for the discovery of service properties facilitates the mapping or *adaptation* of higher-level Grid service functions to such native platform facilities. OGSA's service orientation also allows us to *virtualize* resources at multiple levels, so that the same abstractions and mechanisms can be used both within distributed Grids supporting collaboration across organizational domains and within hosting environments spanning multiple tiers within a single IT domain. A common infrastructure means that differences (e.g., relating to visibility and accessibility) derive from policy controls associated with resource ownership, privacy, and security, rather than interaction mechanisms. Hence, as today's enterprise systems are transformed from separate computing resource islands to integrated, multitiered distributed systems, service components can be integrated dynamically and flexibly, both within and across various organizational boundaries.

The rest of this article is as follows. In Section 2, we examine the issues that motivate the use of Grid technologies in commercial settings. In Section 3, we review the Globus Toolkit and Web services, and in Section 4, we motivate and introduce our Open Grid Services Architecture. In Sections 5–8, we present an example and discuss protocol implementations and higher-level services. We discuss related work in Section 9 and summarize our discussion in Section 10.

We emphasize that the Open Grid Services Architecture and associated Grid service specifications continue to evolve as a result of both standards work within the Global Grid Forum and implementation work within the Globus Project and elsewhere. Thus the technical content in this article, and in an earlier abbreviated presentation [32], represent only a snapshot of a work in progress.

2 The Need for Grid Technologies

Grid technologies support the sharing and coordinated use of diverse resources in dynamic VOs—that is, the creation, from geographically and organizationally distributed components, of virtual computing systems that are sufficiently integrated to deliver desired QoS [34].

Grid concepts and technologies were first developed to enable resource sharing within far-flung scientific collaborations [18, 19, 28, 30, 46, 64]. Applications include collaborative visualization of large scientific datasets (pooling of expertise), distributed computing for computationally demanding data analyses (pooling of compute power and storage), and coupling of scientific instruments with remote computers and archives (increasing functionality as well as availability) [45]. We expect similar applications to become important in commercial settings, initially for scientific and technical computing applications (where we can already point to success stories) and then for commercial distributed computing applications, including enterprise application integration and business to business (B2B) partner collaboration over the Internet. Just as the World Wide Web began as a technology for scientific collaboration and was adopted for e-business, we expect a similar trajectory for Grid technologies.

Nevertheless, we argue that Grid concepts are critically important for commercial computing not primarily as a means of enhancing capability, but rather as a solution to new challenges relating to the construction of reliable, scalable, and secure distributed systems. These challenges derive from the current rush, driven by technology trends and commercial pressures, to decompose and distribute through the network previously monolithic host-centric services, as we now discuss.

2.1 The Evolution of Enterprise Computing

In the past, computing typically was performed within highly integrated host-centric enterprise computing centers. While sophisticated distributed systems (e.g., command and control systems, reservation systems, the Internet Domain Name System [52]) existed, these have remained specialized, niche entities [9, 54].

The rise of the Internet and the emergence of e-business have, however, led to a growing awareness that an enterprise's IT infrastructure also encompasses external networks, resources, and services. Initially, this new source of complexity was treated as a network-centric phenomenon and attempts were made to construct "intelligent networks" that intersect with traditional enterprise IT data centers only at "edge servers": for example, an enterprise's Web point of presence, or the virtual private network server that connects an enterprise network to service provider resources. The assumption was that the impact of e-business and the Internet on an enterprise's core IT infrastructure could thus be managed and circumscribed.

This attempt has, in general, failed because IT services decomposition is also occurring *inside* enterprise IT facilities. New applications are being developed to programming models (such as the Enterprise Java Beans component model [65]) that insulate the application from the underlying computing platform and support portable deployment across multiple platforms. This portability in turn allows platforms to be selected on the basis of price/performance and QoS requirements, rather than operating system supported. Thus, for example, Web serving and caching applications target commodity servers rather than traditional mainframe computing platforms. The resulting proliferation of Unix and NT servers necessitates distributed connections to legacy mainframe application and data assets. Increased load on those assets has caused companies to off-load nonessential functions (such as query processing) from back-end transaction processing systems to mid-tier servers. Meanwhile, Web access to enterprise resources requires ever-faster request servicing, further driving the need to distribute and cache content closer to the edge of the network. The overall result is a decomposition of highly integrated internal IT infrastructure into a collection of heterogeneous and fragmented systems.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

Enterprises must then reintegrate (with QoS) these distributed servers and data resources, addressing issues of navigation, distributed security, and content distribution inside the enterprise, much as on external networks.

In parallel with these developments, enterprises are engaging ever more aggressively in e-business and are realizing that a highly robust IT infrastructure is required to handle the associated unpredictability and rapid growth. Enterprises are also now expanding the scope and scale of their enterprise resource planning projects as they try to provide better integration with customer relationship management, integrated supply chain, and existing core systems. These developments are adding to the significant pressures on the enterprise IT infrastructure.

The aggregate effect is that *qualities of service traditionally associated with mainframe host-centric computing [56] are now essential to the effective conduct of e-business across distributed compute resources, inside as well as outside the enterprise*. For example, enterprises must provide consistent response times to customers, despite workloads with significant deviations between average and peak utilization. Thus, they require flexible resource allocation in accordance with workload demands and priorities. Enterprises must also provide a secure and reliable environment for distributed transactions flowing across a collection of dissimilar servers, must deliver continuous availability as seen by end-users, and must support disaster recovery for business workflow across a distributed network of application and data servers. Yet the current paradigm for delivering QoS to applications via the vertical integration of platform-specific components and services just does not work in today's distributed environment: the decomposition of monolithic IT infrastructures is not consistent with the delivery of QoS through vertical integration of services on a given platform. Nor are distributed resource management capabilities effective, being limited by their proprietary nature, inaccessibility to platform resources, and inconsistencies between similar resources across a distributed environment.

The result of these trends is that IT systems integrators take on the burden of re-integrating distributed compute resources with respect to overall QoS. However, without appropriate infrastructure tools, the management of distributed computing workflow becomes increasingly labor-intensive, complex, and fragile as platform-specific operations staff watch for "fires" in overall availability and performance and verbally collaborate on corrective actions across different platforms. This situation is not scalable, cost-effective, or tenable in the face of changes to the computing environment and application portfolio.

2.2 Service Providers and Business-to-Business Computing

Another key trend is the emergence of service providers (SPs) of various types, such as web-hosting SPs, content distribution SPs, applications SPs, and storage SPs. By exploiting economies of scale, SPs aim to take standard e-business processes, such as creation of a web-portal presence, and provide them to multiple customers with superior price/performance. Even traditional enterprises with their own IT infrastructures are offloading such processes because they are viewed as commodity functions.

Such emerging "eUtilities" (a term used to refer to service providers offering continuous, on-demand access) are beginning to offer a model for carrier-grade IT resource delivery through metered usage and subscription services. Unlike the computing services companies of the past, which tended to provide offline batch-oriented processes, resources provided by eUtilities are often tightly integrated with of enterprise computing infrastructures and used for business processes that span both in-house and outsourced resources. Thus, a price of exploiting the economies of scale that are enabled by eUtility structures is a further decomposition and distribution of enterprise computing functions. EUtilities providers face their own technical challenges. To achieve economies of scale, eUtility providers require server infrastructures that can be easily customized on demand to meet specific customer needs. Thus, there is a demand for

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

IT infrastructure that (1) supports dynamic resource allocation in accordance with service-level agreement policies, efficient sharing and reuse of IT infrastructure at high utilization levels, and distributed security from edge of network to application and data servers and (2) delivers consistent response times and high levels of availability—which in turn drives a need for end-to-end performance monitoring and real-time reconfiguration.

Still another key IT industry trend is cross-enterprise business-to-business (B2B) collaboration such as multi-organization supply chain management, virtual web malls, and electronic market auctions. B2B relationships are, in effect, virtual organizations, as defined above—albeit with particularly stringent requirements for security, auditability, availability, service level agreements, and complex transaction processing flows. Thus, B2B computing represents another source of demand for distributed systems integration, characterized by often large differences among the information technologies deployed within different organizations.

3 Background

We review two technologies on which we build to define the Open Grid Services Architecture: the Globus Toolkit, which has been widely adopted as a Grid technology solution for scientific and technical computing, and Web services, which have emerged as a popular standards-based framework for accessing network applications.

3.1 The Globus Toolkit

The Globus Toolkit [29, 34] is a community-based, open-architecture, open-source set of services and software libraries that support Grids and Grid applications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, fault detection, and portability. Globus Toolkit mechanisms are in use at hundreds of sites and by dozens of major Grid projects worldwide.

The toolkit components that are most relevant to OGSA are the Grid Resource Allocation and Management (GRAM) protocol and its “gatekeeper” service, which provides for secure, reliable, service creation and management [25]; the Meta Directory Service (MDS-2) [24], which provides for information discovery through soft state registration [59, 69], data modeling, and a local registry (“GRAM reporter” [25]); and the Grid Security Infrastructure (GSI), which supports single sign on, delegation, and credential mapping. As illustrated in Figure 1, these components provide the essential elements of a service-oriented architecture, but with less generality than is achieved in OGSA.

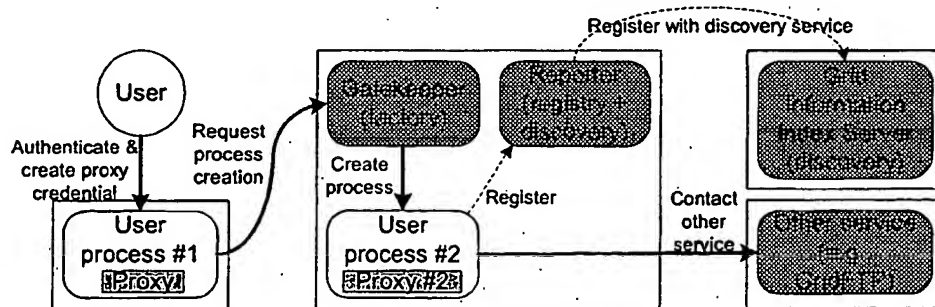


Figure 1: Selected Globus Toolkit mechanisms, showing initial creation of a proxy credential and subsequent authenticated requests to a remote gatekeeper service, resulting in the creation of user process #2, with associated (potentially restricted) proxy credential, followed by a request to another remote service. Also shown is soft-state service registration via MDS-2.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

The GRAM protocol provides for the reliable, secure remote creation and management of arbitrary computations: what we term in this article transient service instances. GSI mechanisms are used for authentication, authorization, and credential delegation [38] to remote computations. A two-phase commit protocol is used for reliable invocation, based on techniques used in the Condor system [50]. Service creation is handled by a small, trusted "gatekeeper" process (termed a factory in this article), while a GRAM reporter monitors and publishes information about the identity and state of local computations (registry).

MDS-2 [24] provides a uniform framework for discovering and accessing system configuration and status information such as compute server configuration, network status, or the locations of replicated datasets (what we term a *discovery* interface in this article). MDS-2 uses a soft-state protocol, the Grid Notification Protocol [44], for lifetime management of published information.

The public-key-based Grid Security Infrastructure (GSI) protocol [33] provides single sign-on authentication, communication protection, and some initial support for restricted delegation. In brief, *single sign-on* allows a user to authenticate once and thus create a proxy credential that a program can use to authenticate with any remote service on the user's behalf. *Delegation* allows for the creation and communication to a remote service of delegated proxy credentials that the remote service can use to act on the user's behalf, perhaps with various restrictions; this capability is important for nested operations. (Similar mechanisms can be implemented within the context of other security technologies, such as Kerberos [63], although with potentially different characteristics.)

GSI uses X.509 certificates, a widely employed standard for PKI certificates, as the basis for user authentication. GSI defines an X.509 proxy certificate [67] to leverage X.509 for support of single sign-on and delegation. (This proxy certificate is similar in concept to a Kerberos forwardable ticket but is based purely on public key cryptographic techniques.) GSI typically uses the Transport Layer Security (TLS) protocol (the follow-on to SSL) for authentication, although other public key-based authentication protocols could be used with X.509 proxy certificates. A remote delegation protocol of X.509 proxy certificates is layered on top of TLS. An Internet Engineering Task Force draft defines the X.509 Proxy Certificate extensions [67]. Global Grid Forum drafts define the delegation protocol for remote creation of an X.509 Proxy Certificate [67] and GSS-API extensions that allow this API to be used effectively for Grid programming.

Rich support for restricted delegation has been demonstrated in prototypes and is a critical part of the proposed X.509 Proxy Certificate Profile [67]. Restricted delegation allows one entity to delegate just a subset of its total privileges to another entity. Such restriction is important to reduce the adverse effects of either intentional or accidental misuse of the delegated credential.

3.2 Web Services

The term *Web services* describes an important emerging distributed computing paradigm that differs from other approaches such as DCE, CORBA, and Java RMI in its focus on simple, Internet-based standards (e.g., eXtensible Markup Language: XML [14, 27]) to address heterogeneous distributed computing. Web services define a technique for describing software components to be accessed, methods for accessing these components, and discovery methods that enable the identification of relevant service providers. Web services are programming language-, programming model-, and system software-neutral.

Web services standards are being defined within the W3C and other standards bodies and form the basis for major new industry initiatives such as Microsoft (.NET), IBM (Dynamic e-Business), and Sun (Sun ONE). We are particularly concerned with three of these standards: SOAP, WSDL, and WS-Inspection.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

- The *Simple Object Access Protocol* (SOAP) [4] provides a means of messaging between a service provider and a service requestor. SOAP is a simple enveloping mechanism for XML payloads that defines a remote procedure call (RPC) convention and a messaging convention. SOAP is independent of the underlying transport protocol; SOAP payloads can be carried on HTTP, FTP, Java Messaging Service (JMS), and the like. We emphasize that Web services can describe multiple access mechanisms to the underlying software component. SOAP is just one means of formatting a Web service invocation.
- The *Web Services Description Language* (WSDL) [22] is an XML document for describing Web services as a set of *endpoints* operating on messages containing either document-oriented (messaging) or RPC payloads. Service interfaces are defined abstractly in terms of message structures and sequences of simple message exchanges (or operations, in WSDL terminology) and then bound to a concrete network protocol and data-encoding format to define an endpoint. Related concrete endpoints are bundled to define abstract endpoints (services). WSDL is extensible to allow description of endpoints and the concrete representation of their messages for a variety of different message formats and network protocols. Several standardized binding conventions are defined describing how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.
- *WS-Inspection* [15] comprises a simple XML language and related conventions for locating service descriptions published by a service provider. A WS-Inspection language (WSIL) document can contain a collection of service descriptions and links to other sources of service descriptions. A service description is usually a URL to a WSDL document; occasionally, a service description can be a reference to an entry within a Universal Description, Discovery, and Integration (UDDI) [5] registry. A link is usually a URL to another WS-Inspection document; occasionally, a link is a reference to a UDDI entry. With WS-Inspection, a service provider creates a WSIL document and makes the document network accessible. Service requestors use standard Web-based access mechanisms (e.g., HTTP GET) to retrieve this document and discover what services the service provider advertises. WSIL documents can also be organized in different forms of index.

Various other Web services standards have been or are being defined. For example, Web Services Flow Language (WSFL) [6] addresses Web services *orchestration*, that is, the building of sophisticated Web services by composing simpler Web services.

The Web services framework has two advantages for our purposes. First, our need to support the dynamic discovery and composition of services in heterogeneous environments necessitates mechanisms for registering and discovering interface definitions and endpoint implementation descriptions and for dynamically generating proxies based on (potentially multiple) bindings for specific interfaces. WSDL supports this requirement by providing a standard mechanism for defining interface definitions separately from their embodiment within a particular binding (transport protocol and data encoding format). Second, the widespread adoption of Web services mechanisms means that a framework based on Web services can exploit numerous tools and extant services, such as WSDL processors that can generate language bindings for a variety of languages (e.g., Web Services Invocation Framework: WSIF [53]), workflow systems that sit on top of WSDL, and hosting environments for Web services (e.g., Microsoft .NET and Apache Axis). We emphasize that the use of Web services does not imply the use of SOAP for all communications. If needed, alternative transports can be used, for example to achieve higher performance or to run over specialized network protocols.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

4. An Open Grid Services Architecture

We have argued that within internal enterprise IT infrastructures, SP-enhanced IT infrastructures, and multi-organizational Grids, computing is increasingly concerned with the creation, management, and application of dynamic ensembles of resources and services (and people)—what we call *virtual organizations* [34]. Depending on context, these ensembles can be small or large, short-lived or long-lived, single institutional or multi-institutional, and homogeneous or heterogeneous. Individual ensembles may be structured hierarchically from smaller systems and may overlap in membership.

We assert that regardless of these differences, developers of applications for VOs face common requirements as they seek to deliver QoS—whether measured in terms of common security semantics, distributed workflow and resource management, coordinated fail-over, problem determination services, or other metrics—across a collection of resources with heterogeneous and often dynamic characteristics.

We now turn to the nature of these requirements and the mechanisms required to address them in practical settings. Extending our analysis in [34], we introduce an Open Grid Services Architecture that supports the creation, maintenance, and application of ensembles of services maintained by VOs.

We start our discussion with some general remarks concerning the utility of a service-oriented Grid architecture, the importance of being able to virtualize Grid services, and essential service characteristics. Then, we introduce the specific aspects that we standardize in our definition of what we call a *Grid service*. We present more technical details in Section 6 (and in [66]).

4.1 Service Orientation and Virtualization

When describing VOs, we can focus on the physical resources being shared (as in [34]) or on the services supported by these resources. (A *service* is a network-enabled entity that provides some capability. The term *object* could arguably also be used, but we avoid that term due to its overloaded meaning.) In OGSA, we focus on *services*: computational resources, storage resources, networks, programs, databases, and the like are all represented as services.

Regardless of our perspective, a critical requirement in a distributed, multiorganizational Grid environment is for mechanisms that enable interoperability [34]. In a service-oriented view, we can partition the interoperability problem into two subproblems, namely the definition of service interfaces and the identification of the protocol(s) that can be used to invoke a particular interface—and, ideally, agreement on a standard set of such protocols.

A service-oriented view allows us to address the need for standard interface definition mechanisms, local/remote transparency, adaptation to local OS services, and uniform service semantics. A service-oriented view also simplifies virtualization—that is, the encapsulation behind a common interface of diverse implementations. Virtualization allows for consistent resource access across multiple heterogeneous platforms with local or remote location transparency, and enables mapping of multiple logical resource instances onto the same physical resource and management of resources within a VO based on composition from lower-level resources. Virtualization allows the composition of services to form more sophisticated services—without regard for how the services being composed are implemented. Virtualization of Grid services also underpins the ability to map common service semantic behavior seamlessly onto native platform facilities.

Virtualization is easier if service functions can be expressed in a standard form, so that any implementation of a service is invoked in the same manner. WSDL, which we adopt for this purpose, supports a service interface *definition* that is distinct from the protocol bindings used for

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

service *invocation*. WSDL allows for multiple bindings for a single interface, including distributed communication protocol(s) (e.g., HTTP) as well as locally optimized binding(s) (e.g., local IPC) for interactions between request and service processes on the same host. Other binding properties may include reliability (and other forms of QoS) as well as authentication and delegation of credentials. The choice of binding should always be transparent to the requestor with respect to service invocation semantics—but not with respect to other things: for example, a requestor should be able to choose a particular binding for performance reasons.

The service interface definition and access binding are also distinct from the *implementation* of the functionality of the service. A service can support multiple implementations on different platforms, facilitating seamless overlay not only to native platform facilities but also, via the nesting of service implementations, to virtual ensembles of resources. Depending on the platform and context, we might use the following implementation approaches.

1. We can use a reference implementation constructed for full portability across multiple platforms to support the execution environment (container) for hosting a service.
2. On a platform possessing specialized native facilities for delivering service functionality, we might map from the service interface definition to the native platform facilities.
3. We can also apply these mechanisms recursively so that a higher-level service is constructed by the composition of multiple lower-level services, which themselves may either map to native facilities or decompose further. The service implementation then dispatches operations to lower-level services (see also Section 4.4)

As an example, consider a distributed trace facility that records trace records to a repository. On a platform that does not support a robust trace facility, a reference implementation can be created and hosted in a service execution environment for storing and retrieving trace records on demand. On a platform already possessing a robust trace facility, however, we can integrate the distributed trace service capability with the native platform trace mechanism, thus leveraging existing operational trace management tools, auxiliary offload, dump/restore, and the like, while semantically preserving the logical trace stream through the distributed trace service. Finally, in the case of a higher-level service, trace records obtained from lower-level services would be combined and presented as the integrated trace facility for the service.

Central to this virtualization of resource behaviors is the ability to adapt to operating system functions on specific hosts. A significant challenge when developing these *mappings* is to enable exploitation of native capabilities—whether concerned with performance monitoring, workload management, problem determination, or enforcement of native platform security policy—so that the Grid environment does not become the least common denominator of its constituent pieces. Grid service discovery mechanisms are important in this regard, allowing higher-level services to discover what capabilities are supported by a particular implementation of an interface. For example, if a native platform supports reservation capabilities, an implementation of a resource management interface (e.g., GRAM [25, 31]) can exploit those capabilities.

Thus, our service architecture supports *local and remote transparency with respect to service location and invocation*. It also provides for *multiple protocol bindings* to facilitate localized optimization of services invocation when the service is hosted locally with the service requestor, as well as to enable protocol negotiation for network flows across organizational boundaries where we may wish to choose between several interGrid protocols, each optimized for a different purpose. Finally, we note that an implementation of a particular Grid service interface may map to native, nondistributed, platform functions and capabilities:

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

4.2 Service Semantics: The Grid Service

Our ability to virtualize and compose services depends on more than standard interface definitions. We also require standard semantics for service interactions so that, for example, different services follow the same conventions for error notification. To this end, OGSA defines what we call a *Grid service*: a Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgradeability. We expect also to address authorization and concurrency control as OGSA evolves. Two other important issues, authentication and reliable invocation, are viewed as service protocol bindings and are thus external to the core Grid service definition, but must be addressed within a complete OGSA implementation. This separation of concerns increases the generality of the architecture without compromising functionality.

The interfaces and conventions that define a Grid service are concerned, in particular, with behaviors related to the management of *transient service instances*. VO participants typically maintain not merely a static set of persistent services that handle complex activity requests from clients. They often need to instantiate new transient service instances dynamically, which then handle the management and interactions associated with the state of particular requested activities. When the activity's state is no longer needed, the service can be destroyed. For example, in a videoconferencing system, the establishment of a videoconferencing session might involve the creation of service instances at intermediate points to manage end-to-end data flows according to QoS constraints. Or, in a Web serving environment, service instances might be instantiated dynamically to provide for consistent user response time by managing application workload through dynamically added capacity. Other examples of transient service instances might be a query against a database, a data mining operation, a network bandwidth allocation, a running data transfer, and an advance reservation for processing capability. (These examples emphasize that service instances can be extremely lightweight entities, created to manage even short-lived activities.) Transience has significant implications for how services are managed, named, discovered, and used.

4.2.1 Upgradeability Conventions and Transport Protocols

Services within a complex distributed system must be independently *upgradeable*. Hence, versioning and compatibility between services must be managed and expressed so that clients can discover not only specific service versions but also compatible services. Further, services (and the hosting environments in which they run) must be upgradeable without disrupting the operation of their clients. For example, an upgrade to the hosting environment may change the set of network protocols that can be used to communicate with the service, and an upgrade to the service itself may correct errors or even enhance the interface. Hence, OGSA defines conventions that allow us to identify when a service changes and when those changes are backwardly compatible with respect to interface and semantics (but not necessarily network protocol). OGSA also defines mechanisms for refreshing a client's knowledge of a service, such as what operations it supports or what network protocols can be used to communicate with the service. A service's description indicates the protocol binding(s) that can be used to communicate with the service. Two properties will often be desirable in such bindings.

- *Reliable service invocation.* Services interact with one another by the exchange of messages. In distributed systems prone to component failure, however, one can never guarantee that a message has been delivered. The existence of internal state makes it important to be able to guarantee that a service has received a message either once or not at all. From this foundation one can build a broad range of higher-level per-operation semantics, such as transactions.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

- **Authentication.** Authentication mechanisms allow the identity of individuals and services to be established for policy enforcement. Thus, one will often desire a transport protocol that provides for mutual authentication of client and service instance, as well as the delegation of proxy credentials. From this foundation one can build a broad range of higher-level authorization mechanisms.

4.2.2 Standard Interfaces

The interfaces (in WSDL terms, portTypes) that define a Grid service are listed in Table 1, introduced here, and described in more detail in Section 6 (and in [66]). Note that while OGSA defines a variety of behaviors and associated interfaces, all but one of these interfaces (*GridService*) are optional.

Table 1: Proposed OGSA Grid service interfaces (see text for details). The names provided here will likely change in the future. Interfaces for authorization, policy management, manageability, and likely other purposes remain to be defined.

PortType	Operation	Description
GridService	FindServiceData	Query a variety of information about the Grid service instance, including basic introspection information (handle, reference, primary key, home handleMap: terms to be defined), richer per-interface information, and service-specific information (e.g., service instances known to a registry). Extensible support for various query languages.
	SetTerminationTime	Set (and get) termination time for Grid service instance
	Destroy	Terminate Grid service instance
Notification-Source	SubscribeTo-NotificationTopic	Subscribe to notifications of service-related events, based on message type and interest statement. Allows for delivery via third party messaging services.
Notification-Sink	DeliverNotification	Carry out asynchronous delivery of notification messages
Registry	RegisterService	Conduct soft-state registration of Grid service handles
	UnregisterService	Deregister a Grid service handle
Factory	CreateService	Create new Grid service instance
HandleMap	FindByHandle	Return Grid Service Reference currently associated with supplied Grid Service Handle

Discovery. Applications require mechanisms for discovering available services and for determining the characteristics of those services so that they can configure themselves and their requests to those services appropriately. We address this requirement by defining

- a standard representation for *service data*, that is, information about Grid service instances, which we structure as a set of named and typed XML elements called *service data elements*, encapsulated in a standard container format;

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

- a standard operation, *FindServiceData* (within the required *GridService* interface), for retrieving service data from individual Grid service instances (“pull” mode access; see the *NotificationSource* interface below for “push” mode access); and
- standard interfaces for registering information about Grid service instances with registry services (*Registry*) and for mapping from “handles” to “references” (*HandleMap*—to be explained in Section 6, when we discuss naming).

Dynamic service creation. The ability to dynamically create and manage new service instances is a basic tenet of the OGSA model and necessitates the existence of service creation services. The OGSA model defines a standard interface (*Factory*) and semantics that any service creation service must provide.

Lifetime management. Any distributed system must be able to deal with inevitable failures. In a system that incorporates transient, stateful service instances, mechanisms must be provided for *reclaiming services and state associated with failed operations*. For example, termination of a videoconferencing session might also require the termination of services created at intermediate points to manage the flow. We address this requirement by defining two standard operations: *Destroy* and *SetTerminationTime* (within the required *GridService* interface), for explicit destruction and soft state lifetime management of Grid service instances, respectively. (*Soft state protocols* [59, 69] allow state established at a remote location to be discarded eventually, unless refreshed by a stream of subsequent “keepalive” messages. Such protocols have the advantages of being both resilient to failure—a single lost message need not cause irretrievable harm—and simple: no reliable “discard” protocol message is required.)

Notification. A collection of dynamic, distributed services must be able to notify each other asynchronously of interesting changes to their state. OGSA defines common abstractions and service interfaces for subscription to (*NotificationSource*) and delivery of (*NotificationSink*) such *notifications*, so that services constructed by the composition of simpler services can deal with notifications (e.g., for errors) in standard ways. The *NotificationSource* interface is integrated with service data, so that a notification request is expressed as a request for subsequent “push” mode delivery of service data. (We might refer to the capabilities provided by these interfaces as an event service [10], but we avoid that term due to its overloaded meaning.)

Other interfaces. We expect to define additional standard interfaces in the near future, to address issues such as authorization, policy management, concurrency control, and the monitoring and management of potentially large sets of Grid service instances.

4.3 The Role of Hosting Environments

OGSA defines the semantics of a Grid service instance: how it is created, how it is named, how its lifetime is determined, how to communicate with it, and so on. However, while OGSA is prescriptive on matters of basic behavior, it does not place requirements on what a service does or how it performs that service. In other words, OGSA does not address issues of implementation programming model, programming language, implementation tools, or execution environment.

In practice, Grid services are instantiated within a specific execution environment or *hosting environment*. A particular hosting environment defines not only implementation programming model, programming language, development tools, and debugging tools, but also how an implementation of a Grid service meets its obligations with respect to Grid service semantics.

Today’s e-science Grid applications typically rely on *native operating system processes* as their hosting environment, with for example creation of a new service instance involving the creation of a new process. In such environments, a service itself may be implemented in a variety of languages such as C, C++, Java, or Fortran. Grid semantics may be implemented directly as part

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

of the service, or provided via a linked library [39]. Typically semantics are not provided via external services, beyond those provided by the operating system. Thus, for example, lifetime management functions must be addressed within the application itself, if required.

Web services, on the other hand, may be implemented on more sophisticated *container or component-based* hosting environments such as J2EE, Websphere, .NET, and Sun One. Such environments define a framework (container) within which components adhering to environment-defined interface standards can be instantiated and composed to build complex applications. Compared with the low levels of functionality provided by native hosting environments, container/component hosting environments tend to offer superior programmability, manageability, flexibility, and safety. Consequently, component/container based hosting environments are seeing widespread use for building e-business services. In the OGSA context, the container (hosting environment) has primary responsibility for ensuring that the services it supports adhere to Grid service semantics, and thus OGSA may motivate modifications or additions to the container/component interface.

By defining service semantics, OGSA specifies interactions between services in a manner independent of any hosting environment. However, as the above discussion highlights, successful implementation of Grid services can be facilitated by specifying baseline characteristics that all hosting environments must possess, defining the "internal" interface from the service implementation to the global Grid environment. These characteristics would then be rendered into different implementation technologies (e.g., J2EE or shared libraries).

A detailed discussion of hosting environment characteristics is beyond the scope of this article. However, we can expect a hosting environment to address mapping of Grid-wide names (i.e., Grid service handles) into implementation-specific entities (C pointers, Java object references, etc.); dispatch of Grid invocations and notification events into implementation-specific actions (events, procedure calls); protocol processing and the formatting of data for network transmission; lifetime management of Grid service instances; and inter-service authentication.

4.4 Using OGSA Mechanisms to Build VO Structures

Applications and users must be able to create transient services and to discover and determine the properties of available services. The OGSA *Factory*, *Registry*, *GridService*, and *HandleMap* interfaces support the creation of transient service instances and the discovery and characterization of the service instances associated with a VO. (In effect, a registry service—a service instance that supports the *Registry* interface for registration and the *GridService* interface's *FindServiceData* operation, with appropriate service data, for discovery—defines the service set associated with a VO.) These interfaces can be used to construct a variety of VO service structures, as illustrated in Figure 2 and described in the following.

Simple hosting environment. A simple execution environment is a set of resources located within a single administrative domain and supporting native facilities for service management: for example, a J2EE application server, Microsoft .NET system, or Linux cluster. In OGSA, the user interface to such an environment will typically be structured as a registry, one or more factories, and a handleMap service. Each factory is recorded in the registry, to enable clients to discover available factories. When a factory receives a client request to create a Grid service instance, the factory invokes hosting-environment-specific capabilities to create the new instance, assigns it a handle, registers the instance with the registry, and makes the handle available to the handleMap service. The implementations of these various services map directly into local operations.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

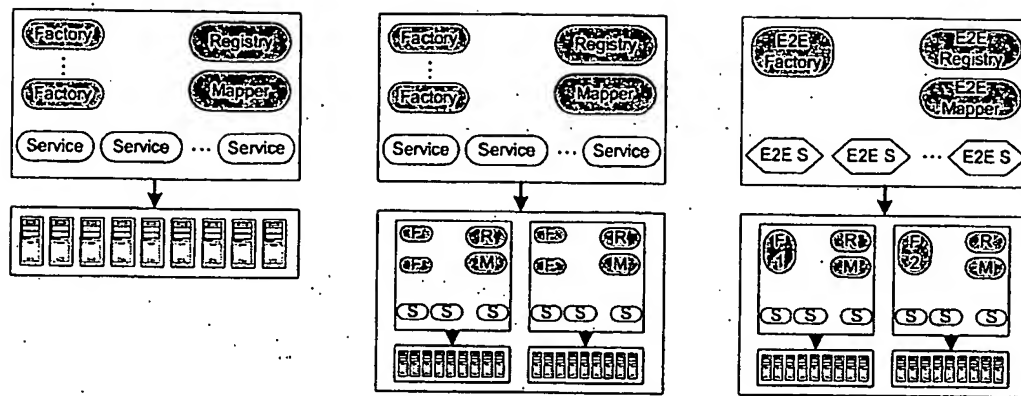


Figure 2: Three different VO structures, as described in the text. From left to right: simple hosting environment, virtual hosting environment, and collective services.

Virtual hosting environment. In more complex environments, the resources associated with a VO will span heterogeneous, geographically distributed “hosting environments.” (For example, in Figure 2, these resources span two simple hosting environments.) Nevertheless, this “virtual hosting environment” (which corresponds, perhaps, to the set of resources associated with a B2B partnership) can be made accessible to a client via exactly the same interfaces as were used for the hosting environment just described. We create one or more “higher-level” factories that delegate creation requests to lower-level factories. Similarly, we create a higher-level registry that knows about the higher-level factories and the service instances that they have created, as well as any VO-specific policies that govern the use of VO services. Clients can then use the VO registry to find factories and other service instances associated with the VO, and then use the handles returned by the registry to talk directly to those service instances. The higher-level factories and registry implement standard interfaces and so, from the perspective of the user, are indistinguishable from any other factory or registry.

Note that here, as in the previous example, the registry handle can be used as a globally unique name for the service set maintained by the VO. Resource management policies can be defined and enforced on the platforms hosting VO services, targeting the VO by this unique name.

Collective operations. We can also construct a “virtual hosting environment” that provides VO participants with more sophisticated, virtual, “collective” or “end-to-end” services. In this case, the registry keeps track of and advertises factories that create higher-level service instances. Such instances are implemented by asking lower-level factories to create multiple service instances and by composing the behaviors of those multiple lower-level service instances into that single, higher-level service instance.

These three examples, and the preceding discussion, illustrate how Grid service mechanisms can be used to integrate distributed resources both across virtual multi-organizational boundaries and within internal commercial IT infrastructures. In both cases, a collection of Grid services registered with appropriate discovery services can support functional capabilities delivering QoS interactions across distributed resource pools. Applications and middleware can exploit these services for distributed resource management across heterogeneous platforms with local and remote transparency and locally optimized flows.

Implementations of Grid services that map to native platform resources and APIs enable seamless integration of higher-level Grid services such as those just described with underlying platform components. Furthermore, service sets associated with multiple VOs can map to the same

underlying physical resources, with those services represented as logically distinct at one level but sharing physical resource systems at lower levels.

5 Application Example

We illustrate in Figure 3 the following stages in the life of a data mining computation, which we use to illustrate the working of basic remote service invocation, lifetime management, and notification functions.

1. The environment initially comprises (from left to right) four simple hosting environments: one that runs the user application; one that encapsulates computing and storage resources (and that supports two factory services, one for creating storage reservations and the other for creating mining services); and two that encapsulate database services. The "R"s represent local registry services; an additional VO registry service presumably provides information about the location of all depicted services.
2. The user application invokes "create Grid service" requests on the two factories in the second hosting environment, requesting the creation of a "data mining service" that will perform the data mining operation on its behalf, and an allocation of temporary storage for use by that computation. Each request involves mutual authentication of the user and the relevant factory (using an authentication mechanism described in the factory's service description) followed by authorization of the request. Each request is successful and results in the creation of a Grid service instance with some initial lifetime. The new data mining service instance is also provided with delegated proxy credentials that allow it to perform further remote operations on behalf of the user.
3. The newly created data mining service uses its proxy credentials to start requesting data from the two database services, placing intermediate results in local storage. The data mining service also uses notification mechanisms to provide the user application with periodic updates on its status. Meanwhile, the user application generates periodic "keepalive" requests to the two Grid service instances that it has created.
4. The user application fails for some reason. The data mining computation continues for now, but as no other party has an interest in its results, no further keepalive messages are generated.
5. (Not shown in figure) Due to the application failure, keepalive messages cease, and so the two Grid service instances eventually time out and are terminated, freeing the storage and computing resources that they were consuming.

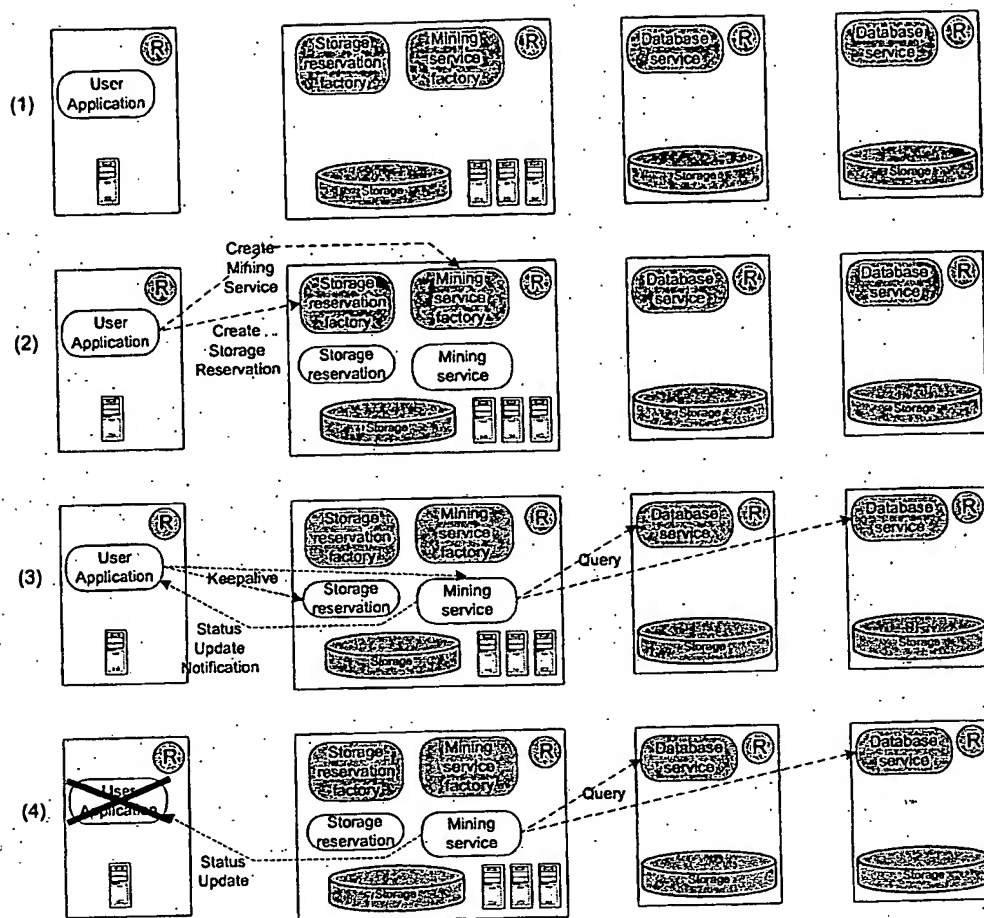


Figure 3: An example of Grid services at work. See text for details.

6 Technical Details

We now present a more detailed description of the Grid service abstraction and associated interfaces and conventions.

6.1 The OGSA Service Model

A basic premise of OGSA is that everything is represented by a *service*: a network enabled entity that provides some capability through the exchange of messages. Computational resources, storage resources, networks, programs, databases, and so forth are all services. This adoption of a uniform service-oriented model means that all components of the environment are virtual.

More specifically, OGSA represents everything as a *Grid service*: a Web service that conforms to a set of conventions and supports standard interfaces for such purposes as lifetime management. This core set of consistent interfaces, from which all Grid services are implemented, facilitates the construction of higher-order services that can be treated in a uniform way across layers of abstraction.

This is a **DRAFT** document and a work in progress. Version: 6/22/2002.
 Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

Grid services are characterized (*typed*) by the capabilities that they offer. A Grid service implements one or more *interfaces*, where each interface defines a set of operations that are invoked by exchanging a defined sequence of messages. Grid service interfaces correspond to *portTypes* in WSDL. The set of *portTypes* supported by a Grid service, along with some additional information relating to versioning, are specified in the Grid service's *serviceType*, a WSDL extensibility element defined by OGSA.

Grid services can maintain internal state for the lifetime of the service. The existence of state distinguishes one *instance* of a service from another that provides the same interface. We use the term *Grid service instance* to refer to a particular instantiation of a Grid service.

The protocol binding associated with a service interface can define a delivery semantics that addresses, for example, reliability. Services interact with one another by the exchange of messages. In distributed systems prone to component failure, however, one can never guarantee that a message that is sent has been delivered. The existence of internal state can make it important to be able to guarantee that a service has received a message once or not at all, even if failure recovery mechanisms such as retry are in use. In such situations, we may wish to use a protocol that guarantees exactly-once delivery or some similar semantics. Another frequently desirable protocol binding behavior is mutual authentication during communication.

OGSA services can be created and destroyed dynamically. Services may be destroyed explicitly, or may be destroyed or become inaccessible as a result of some system failure such as operating system crash or network partition. Interfaces are defined for managing service lifetime.

Because Grid services are dynamic and stateful, we need a way to distinguish one dynamically created service instance from another. Thus, every Grid service instance is assigned a globally unique name, the *Grid service handle (GSH)*, that distinguishes a specific Grid service instance from all other Grid service instances that have existed, exist now, or will exist in the future. (If a Grid service fails and is restarted in such a way as to preserve its state, then it is essentially the same instance, and the same GSH can be used.)

Grid services may be upgraded during their lifetime, for example to support new protocol versions or to add alternative protocols. Thus, the GSH carries no protocol- or instance-specific information such as network address and supported protocol bindings. Instead, this information is encapsulated, along with all other instance-specific information required to interact with a specific service instance, into a single abstraction called a *Grid service reference (GSR)*. Unlike a GSH, which is invariant, the GSR(s) for a Grid service instance can change over that service's lifetime. A GSR has an explicit expiration time, or may become invalid at any time during a service's lifetime, and OGSA defines mapping mechanisms, described below, for obtaining an updated GSR.

The result of using a GSR whose lifetime has expired is undefined. Note that holding a valid GSR does not guarantee access to a Grid service instance: local policy or access control constraints (for example maximum number of current requests) may prohibit servicing a request. In addition, the referenced Grid service instance may have failed, preventing the use of the GSR.

As everything in OGSA is a Grid service, there must be Grid services that manipulate the Grid service, handle, and reference abstractions that define the OGSA model. Defining a specific set of services would result in a specific rendering of the OGSA service model. We therefore take a more flexible approach and define a set of basic OGSA interfaces (i.e., WSDL *portTypes*) for manipulating service model abstractions. These interfaces can then be combined in different ways to produce a rich range of Grid services. Table 1 presents names and descriptions for the Grid service interfaces defined to date. Note that *only the GridService interface must be supported by all Grid services*.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

6.2 Creating Transient Services: Factories

OGSA defines a class of Grid services that implement an interface that creates new Grid service instances. We call this the *Factory* interface and a service that implements this interface a *factory*. The *Factory* interface's *CreateService* operation creates a requested Grid service and returns the GSH and initial GSR for the new service instance.

The *Factory* interface does not specify how the service instance is created. One common scenario is for the factory interface to be implemented in some form of hosting environment (such as .NET or J2EE) that provides standard mechanisms for creating (and subsequently managing) new service instances. The hosting environment may define how services are implemented (e.g., language), but this is transparent to service requestors in OGSA, which see only the factory interface. Alternatively, one can construct "higher-level" factories that create services by delegating the request to other factory services (see Section 4.4). For example, in a Web serving environment, a new computer might be integrated into the active pool by asking an appropriate factory service to instantiate a "Web serving" service on an idle computer.

6.3 Service Lifetime Management

The introduction of transient service instances raises the issue of determining the service's lifetime: that is, determining when a service can or should be terminated so that associated resources can be recovered. In normal operating conditions, a transient service instance is created to perform a specific task and either terminates on completion of this task or via an explicit request from the requestor or from another service designated by the requestor. In distributed systems, however, components may fail and messages may be lost. One result is that a service may never see an expected explicit termination request, thus causing it to consume resources indefinitely.

OGSA addresses this problem through a soft state approach [23, 69] in which Grid service instances are created with a specified lifetime. The initial lifetime can be extended by a specified time period by explicit request of the client or another Grid service acting on the client's behalf (subject of course to the policy of the service). If that time period expires without having received a re-affirmation of interest from a client, either the hosting environment or the service instance itself is at liberty to terminate the service instance and release any associated resources.

Our approach to Grid service lifetime management has two desirable properties:

- A client knows, or can determine, when a Grid service instance will terminate. This knowledge allows the client to determine reliably when a service instance has terminated and hence its resources have been recovered, even in the face of system faults (e.g., failures of servers, networks, clients). The client knows exactly how long it has in order to request a final status from the service instance or to request an extension to the service's lifetime. Moreover, it also knows that if system faults occur, it need not continue attempting to contact a service after a known termination time, and that any resources associated with that service would be released after that time—unless another client succeeded in extending the lifetime. In brief, lifetime management enables robust termination and failure detection, by clearly defining the lifetime semantics of a service instance.
- A hosting environment is guaranteed that resource consumption is bounded, even in the face of system failures outside of its control. If the termination time of a service is reached, the hosting environment can reclaim all associated resources.

We implement soft state lifetime management via the *SetTerminationTime* operation within the required *GridService* interface, which defines operations for negotiating an initial lifetime for a

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

new service instance, for requesting a lifetime extension, and for harvesting a service instance when its lifetime has expired. We describe each of these mechanisms in turn.

Negotiating an initial lifetime. When requesting the creation of a new Grid service instance through a factory, a client indicates minimum and maximum acceptable initial lifetimes. The factory selects an initial lifetime and returns this to the client.

Requesting a lifetime extension. A client requests a lifetime extension via a *SetTerminationTime* message to the Grid service instance, which specifies a minimum and maximum acceptable new lifetime. The service instance selects a new lifetime and returns this to the client. Note that a keepalive message is effectively idempotent: the result of a sequence of requests is the same, even if intermediate requests are lost or reordered, as long as not so many requests are lost that the service instance's lifetime expires.

The periodicity of keepalive messages can be determined by the client based on the initial lifetime negotiated with the service instance (and perhaps renegotiated via subsequent keepalive messages) and knowledge about network reliability. The interval size allows tradeoffs between currency of information and overhead.

We note that this approach to lifetime management provides a service with considerable autonomy. Lifetime extension requests from clients are not mandatory: the service can apply its own policies on granting such request. A service can decide at any time to extend its lifetime, either in response to a lifetime extension request by a client or any other reason. A service instance can also cancel itself at any time, for example if resource constraints and priorities dictate that it relinquishes its resources. Subsequent client requests that refer to this service will fail.

The use of absolute time in the *SetTerminationTime* operation—and, for that matter, in Grid service information elements, and commonly in security credentials—implies the existence of a global clock that is sufficiently well synchronized. The Network Time Protocol (NTP) provides standardized mechanisms for clock synchronization and can typically synchronize clocks within at most tens of milliseconds, which is more than adequate for the purposes of lifetime management. Note that we are not implying by these statements a requirement for ordering of events, although we expect to introduce some such mechanisms in future revisions.

6.4 Managing Handles and References

As discussed above, the result of a factory request is a GSH and a GSR. While the GSH is guaranteed to reference the created Grid service instance in perpetuity, the GSR is created with a finite lifetime and may change during the service's lifetime. While this strategy has the advantage of increased flexibility from the perspective of the Grid service provider, it introduces the problem of obtaining a valid GSR once the GSR returned by the service creation operation expires. At its core, this is a bootstrapping problem: how does one establish communication with a Grid service given only its GSH? We describe here how these issues are addressed in the Grid service specification as of June 2002, but note that this part of the specification is likely to evolve in the future, at a minimum to support multiple handle representations and handle mapping services.

The approach taken in OGSA is to define a handle-to-reference mapper interface (*HandleMap*). The operations provided by this interface take a GSH and return a valid GSR. Mapping operations can be access controlled and thus a mapping request may be denied. An implementation of the *HandleMap* interface may wish to keep track of what Grid service instances are actually in existence and not return references to instances that it knows have terminated. However, possession of a valid GSR does not assure that a Grid service instance can be contacted: the service may have failed or been explicitly terminated between the time the GSR

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

was given out and the time that it was used. (Obviously, if termination of a service is scheduled, it is desirable to represent this in the GSR lifetime, but it is not required.)

By introducing the *HandleMap* interface, we partition the general problem of obtaining a GSR for an arbitrary service into two more specific subproblems:

- 1) Identifying a *handleMap* service that contains the mapping for the specified GSH, and
- 2) Contacting that *handleMap* to obtain the desired GSR.

We address these two subproblems in turn. To ensure that we can always map a GSH to a GSR, we require that every Grid service instance be registered with at least one *handleMap*, which we call the *home handleMap*. By structuring the GSH to include the *home handleMap*'s identity, we can easily and scalably determine which *handleMap* to contact to obtain a GSR for a given GSH. Hence, unique names can be determined locally, thus avoiding scalability problems associated with centralized name allocation services—although relying on the Domain Name System [52]. Note that GSH mappings can also live in other *handleMaps*. However, every GSH must have exactly one *home handleMap*.

How do we identify the *home handleMap* within a GSH? Any service that implements the *HandleMap* interface is a Grid service, and as such will have a GSH. If we use this name in constructing a GSH, however, then we are back in the same position of trying to obtain a GSR from the *handleMap* service's GSH. To resolve this bootstrapping problem, we need a way to obtain the GSR for the *handleMap* without requiring a *handleMap*! We accomplish this by requiring that all *home handleMap* services be identified by a URL and support a bootstrapping operation that is bound to a single, well-known protocol, namely, HTTP (or HTTPS). Hence, instead of using a GSR to describe what protocols should be used to contact the *handleMap* service, an HTTP GET operation is used on the URL that points to the *home handleMap*, and the GSR for the *handleMap*, in WSDL form, is returned.

Note that a relationship exists between services that implement the *HandleMap* and *Factory* interfaces. Specifically, the GSH returned by a factory request must contain the URL of the *home handleMap*, and the GSH/GSR mapping must be entered and updated into the *handleMap* service. The implementation of a factory must decide what service to use as the *home handleMap*. Indeed a single service may implement both the *Factory* and *HandleMap* interfaces.

Current work within GGF is revising this Grid service component to allow for other forms of handles and mappers/resolvers and/or to simplify the current handle and resolver.

6.5 Service Data and Service Discovery

Associated with each Grid service instance is a set of *service data*, a collection of XML elements encapsulated as service data elements. The packaging of each element includes a name that is unique to the Grid service instance, a type, and time-to-live information that a recipient can use for lifetime management.

The obligatory *GridService* interface defines a standard WSDL operation, *FindServiceData*, for querying and retrieving service data. This operation requires a simple "by name" query language, and is extensible to allow for the specification of the query language used, which may be for example Xquery [20].

The Grid service specification defines for each Grid service interface a set of zero or more service data elements that must be supported by any Grid service instance that supports that interface. Associated with the *GridService* interface, and thus obligatory for any Grid service instance, are a set of elements containing basic information about a Grid service instance, such as its GSH, GSR, primary key, and *home handleMap*.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov; carl@isi.edu; jnick@us.ibm.com; tuecke@mcs.anl.gov

One application of the *GridService* interface's *FindServiceData* operation is service discovery. Our discussion above assumed that one has a GSH that represents a desired service. But how does one obtain the GSH in the first place? This is the essence of *service discovery*, which we define here as the process of identifying a subset of GSHs from a specified set based on GSH attributes such as the interfaces provided, the number of requests that have been serviced, the load on the service, or policy statements such as the number of outstanding requests allowed.

A Grid service that supports service discovery is called a *registry*. A registry service is defined by two things: the *Registry* interface, which provides operations by which GSHs can be registered with the registry service, and an associated service data element used to contain information about registered GSHs. Thus, the *Registry* interface is used to register a GSH and the *GridService* interface's *FindServiceData* operation is used to retrieve information about registered GSHs.

The *Registry* interface allows a GSH to register with a registry service to augment the set of GSHs that are considered for subsetting. As in MDS-2 [24], a service (or VO) can use this operation to notify interested parties within a VO of its existence and the service(s) that it provides. These interested parties typically include various forms of service discovery services, which collect and structure service information in order to respond efficiently to service discovery requests. As with other stateful interfaces in OGSA, GSH registration is a soft state operation and must be periodically refreshed, thus allowing discovery services to deal naturally with dynamic service availability.

We note that specification of the attributes associated with a GSH is not tied to the registration of a GSH to a service implementing the *GridService* interface. This feature is important because attribute values may be dynamic and there may be a variety of ways in which attribute values may be obtained, including consulting another service implementing the *GridService* interface.

6.6 Notification

The OGSA notification framework allows clients to register interest in being notified of particular messages (the *NotificationSource* interface) and supports asynchronous, one-way delivery of such notifications (*NotificationSink*). If a particular service wishes to support subscription of notification messages, it must support the *NotificationSource* interface to manage the subscriptions. A service that wishes to receive notification messages must implement the *NotificationSink* interface, which is used to deliver notification messages. To start notification from a particular service, one invokes the *subscribe* operation on the notification source interface, giving it the service GSH of the notification sink. A stream of notification messages then flow from the source to the sink, while the sink sends periodic keepalive messages to notify the source that it is still interested in receiving notifications. If reliable delivery is desired, this behavior can be implemented by defining an appropriate protocol binding for this service.

An important aspect of this notification model is a close integration with service data: a subscription operation is just a request for subsequent "push" delivery of service data that meet specified conditions. (Recall that the *FindServiceData* operation provides a "pull" model.)

The framework allows both for direct service-to-service notification message delivery, and for integration with various third-party services, such as messaging services commonly used in the commercial world, or custom services that filter, transform, or specially deliver notification messages on behalf of the notification source. Notification semantics are a property of the protocol binding used to deliver the message. For example, a SOAP/HTTP protocol or direct UDP binding would provide point-to-point, best-effort, notification, while other bindings (e.g., some proprietary message service) would provide better than best-effort delivery. A multicast protocol binding would support multiple receivers.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

6.7 Change Management

In order to support *discovery* and *change management* of Grid services, Grid service interfaces must be globally and uniquely named. In WSDL, an interface is defined by a portType and is globally and uniquely named by the portType's QName (i.e., an XML namespace as defined by the targetNamespace attribute in the WSDL document's definitions element, and a local name defined by the portType element's name attribute). Any changes made to the definition of a Grid service, either by changing its interface or by making semantically significant implementation changes to the operations, must be reflected through new interface names (i.e., new portTypes and/or serviceTypes). This feature allows clients that require Grid Services with particular properties (either particular interfaces or implementation semantics) to discover compatible services.

6.8 Other Interfaces

We expect in the future to define an optional *Manageability* interface that supports a set of manageability operations. Such operations allow potentially large sets of Grid service instances to be monitored and managed from management consoles, automation tools, and the like. An optional *Concurrency* interface will provide concurrency control operations.

7 Network Protocol Bindings

The Web services framework can be instantiated on a variety of different protocol bindings. SOAP+HTTP with TLS for security is one example, but others can and have been defined. Here we discuss some issues that arise in the OGSA context.

In selecting network protocol bindings within an OGSA context, we must address four primary requirements:

- *Reliable transport.* As discussed above, the Grid services abstraction can require support for reliable service invocation. One way to address this requirement is to incorporate appropriate support within the network protocol binding, as for example in HTTP-R.
- *Authentication and delegation.* As discussed above, the Grid services abstraction can require support for communication of proxy credentials to remote sites. One way to address this requirement is to incorporate appropriate support within the network protocol binding, as for example in TLS extended with proxy credential support.
- *Ubiquity.* The Grid goal of enabling the dynamic formation of VOs from distributed resources means that, in principle, it must be possible for any arbitrary pair of services to interact.
- *GSR Format.* Recall that the Grid Service Reference can take a binding-specific format. One possible GSR format is a WSDL document; CORBA IOR is another.

The successful deployment of large-scale interoperable OGSA implementations would benefit from the definition of a small number of standard protocol bindings for Grid service discovery and invocation. Just as the ubiquitous deployment of the Internet Protocol allows essentially any two entities to communicate, so ubiquitous deployment of such "InterGrid" protocols will allow any two services to communicate. Hence, clients can be particularly simple, since they need to know about only one set of protocols. (Notice that the definition of such standard protocols does not prevent a pair of services from using an alternative protocol, if both support it.) Whether or not such InterGrid protocols can be defined and gain widespread acceptance remains to be seen. In any case, their definition is beyond the scope of this article.

8 Higher-Level Services

The abstractions and services described in this article provide building blocks that can be used to implement a variety of higher-level Grid services. We intend to work closely with the community to define and implement a wide variety of such services that will, collectively, address the diverse requirements of e-business and e-science applications. These are likely to include the following:

- *Distributed data management services*, supporting access to and manipulation of distributed data, whether in databases or files [58]. Services of interest include database access, data translation, replica management, replica location, and transactions.
- *Workflow services*, supporting the coordinated execution of multiple application tasks on multiple distributed Grid resources.
- *Auditing services*, supporting the recording of usage data, secure storage of that data, analysis of that data for purposes of fraud and intrusion detection, and so forth.
- *Instrumentation and monitoring services*, supporting the discovery of "sensors" in a distributed environment, the collection and analysis of information from these sensors, the generation of alerts when unusual conditions are detected, and so forth.
- *Problem determination services for distributed computing*, including dump, trace, and log mechanisms with event tagging and correlation capabilities.
- *Security protocol mapping services*, enabling distributed security protocols to be transparently mapped onto native platform security services for participation by platform resource managers not implemented to support the distributed security authentication and access control mechanism.

The flexibility of our framework means that such services can be implemented and composed in a variety of different ways. For example, a coordination service that supports the simultaneous allocation and use of multiple computational resources can be instantiated as a service instance, linked with an application as a library, or incorporated into yet higher-level services.

It appears straightforward to re-engineer the resource management, data transfer, and information service protocols used within the current Globus Toolkit to build on these common mechanisms (see Figure 4). In effect, we can *refactor* the design of those protocols, extracting similar elements to exploit commonalities. In the process, we enhance the capabilities of the current protocols and arrive at a common service infrastructure. This process will produce Globus Toolkit 3.0.

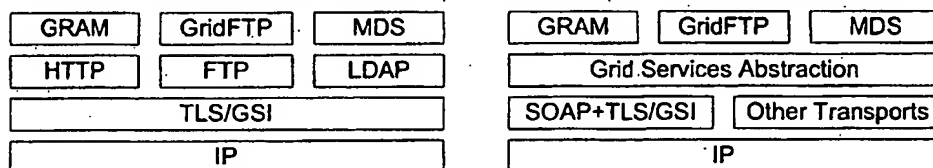


Figure 4: On the left, some current Globus Toolkit protocols; on the right, a potential refactoring to exploit OGSA mechanisms.

9 Related Work

We note briefly some relevant prior and other related work, focusing in particular on issues relating to the secure and reliable remote creation and management of transient, stateful services.

As discussed in Section 3.1, many OGSA mechanisms derive from the Globus Toolkit v2.0: in particular, the factory (GRAM gatekeeper [25]), registry (GRAM reporter [25] and MDS-2 [24]),

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

use of soft-state registration (MDS-2 [24]), secure remote invocation with delegation (GSI [33]), and reliable remote invocation (GRAM [25]). The primary differences relate to how these different mechanisms are integrated, with OGSA refactoring key design elements so that, for example, common notification mechanisms are used for service registration and service state.

OGSA can be viewed as a distributed object system [21], in the sense that each Grid service instance has a unique identity with respect to the other instances in the system, and each instance can be characterized as state coupled with behavior published through type-specific operations. In this respect, OGSA exploits ideas developed previously in systems such as Eden [7], Argus [48], CORBA [1], SOS [60], Spring [51], Globe [62], Mentat [42], and Legion [41, 43], among others. In contrast to CORBA, OGSA like Web services addresses directly issues of secure interoperability and provides a richer interface definition language. In Grid computing, the Legion group has promoted the use of object models, and we can draw parallels between certain OGSA and Legion constructs, in particular the factory ("Class Object"), handleMap ("Binding Agent"), and timeouts on bindings. However, we also note that OGSA is nonprescriptive on several issues that are often viewed as central to distributed object systems, such as the use of object technologies in implementations, the exposure of inheritance mechanisms at the interface level, and hosting technology.

Soft state mechanisms have been used for management of specific state in network entities within Internet protocols [23, 61, 69] and (under the name "leases") in RMI and Jini [57]. In OGSA, all services and information are open to soft state management. We prefer soft state techniques to alternatives such as distributed reference counting [12] because of their relative simplicity.

Our reliable invocation mechanisms are inspired by those used in Condor [36, 49, 50], which in turn build on much prior work in distributed systems.

As noted in Section 4.3, core OGSA service behaviors will, in general, be supported via some form of hosting environment that simplifies the development of individual components by managing persistence, security, lifecycle management, and so forth. The notion of a hosting environment appears in various operating systems and object systems.

The application of Web services mechanisms to Grid computing has also been investigated and advocated by others (e.g., [35, 37]), with a recent workshop providing overviews of a number of relevant efforts [2]. Gannon et al. [37] discuss the application of various contemporary technologies to e-science applications and propose "application factories" (with WSDL interfaces) as a means of creating application services dynamically. De Roure et al. [26] propose a "Semantic Grid," by analogy to the Semantic Web [11], and propose a range of higher-level services. Work on service-oriented interfaces to numerical software in NetSolve [16, 17] and Ninf [55] is also relevant.

Sun Microsystems' JXTA system [3] addresses several important issues encountered in Grids, including discovery of, and membership in, virtual organizations—what JXTA calls "peer groups." We believe that these abstractions can be implemented within the OGSA framework.

There are connections to be made with component models for distributed and high-performance computing [8, 13, 68], some implementations of which build on Globus Toolkit mechanisms.

10 Summary

We have defined an Open Grid Services Architecture (OGSA) that supports, via standard interfaces and conventions, the creation, termination, management, and invocation of *stateful, transient services as named, managed entities with dynamic, managed lifetime*.

Within OGSA, everything is represented as a *Grid service*, that is, a (potentially transient) service that conforms to a set of conventions (expressed using WSDL) for such purposes as lifetime management, discovery of characteristics, notification, and so on. Grid service implementations can target native platform facilities for integration with, and of, existing IT infrastructures. Standard interfaces for creating, registering, and discovering Grid services can be configured to create various forms of VO structure.

The merits of this service-oriented model are as follows. All components of the environment are virtualized. By providing a core set of consistent interfaces from which all Grid services are implemented, we facilitate the construction of hierarchal, higher-order services that can be treated in a uniform way across layers of abstraction. Virtualization also enables mapping of multiple logical resource instances onto the same physical resource, composition of services regardless of implementation, and management of resources within a VO based on composition from lower-level resources. It is virtualization of Grid services that underpins the ability for mapping common service semantic behavior seamlessly onto native platform facilities.

The development of OGSA represents a natural evolution of the Globus Toolkit 2.0, in which the key concepts of factory, registry, reliable and secure invocation, etc., exist, but in a less general and flexible form than here, and without the benefits of a uniform interface definition language. In effect, OGSA refactors key design elements so that, for example, common notification mechanisms are used for service registration and service state. OGSA also further abstracts these elements so that they can be applied at any level to virtualize VO resources. The Globus Toolkit provides the basis for an open source OGSA implementation, Globus Toolkit 3.0, that supports existing Globus APIs as well as WSDL interfaces, as described at www.globus.org/ogsa.

The development of OGSA also represents a natural evolution of Web services. By integrating support for transient, stateful service instances with existing Web services technologies, OGSA extends significantly the power of the Web services framework, while requiring only minor extensions to existing technologies.

Acknowledgments

We are pleased to acknowledge the many contributions to the Open Grid Services Architecture of Karl Czajkowski, Jeffrey Frey, and Steve Graham. We are also grateful to numerous colleagues for discussions on the topics covered here and/or for helpful comments on versions of this article, in particular Malcolm Atkinson, Brian Carpenter, David De Roure, Andrew Grimshaw, Marty Humphrey, Keith Jackson, Bill Johnston, Kate Keahey, Gregor von Laszewski, Lee Liming, Miron Livny, Norman Paton, Jean-Pierre Prost, Thomas Sandholm, Peter Vanderbilt, and Von Welch.

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the National Science Foundation; by the NASA Information Power Grid program; and by IBM.

Bibliography

1. Common Object Request Broker: Architecture and Specification, Revision 2.2. Object Management Group Document 96.03.04, 1998.
2. Grid Web Services Workshop. 2001, <https://gridport.npaci.edu/workshop/webserv01/agenda.html>.
3. JXTA. www.jxta.org.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

4. Simple Object Access Protocol (SOAP) 1.1. W3C, Note 8, 2000.
5. UDDI: Universal Description, Discovery and Integration. www.uddi.org.
6. Web Services Flow Language. www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
7. Almes, G.T., Black, A.P., Lazowska, E.D. and Noe, J.D. The Eden System: A Technical Review. *IEEE Transactions on Software Engineering*, SE-11 (1). 43--59. 1985.
8. Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L. and Parker, S. Toward a Common Component Architecture for High Performance Scientific Computing. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*, 1999.
9. Bal, H.E., Steiner, J.G. and Tanenbaum, A.S. Programming Languages for Distributed Computing Systems. *ACM Computing Surveys*, 21 (3). 261--322. 1989.
10. Barrett, D.J., Clarke, L.A., Tarr, P.L. and Wise, A.E. A Framework for Event-based Software Integration. *ACM Transactions on Software Engineering and Methodology*, 5 (4). 378-421. 1996.
11. Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. *Scientific American*. 2001.
12. Bevan, D.I., Distributed Garbage Collection Using Reference Counting. In *PARLE Parallel Architectures and Languages Europe*, (1987), Springer Verlag, LNCS 259, 176-187.
13. Bramley, R., Gannon, D., Stuckey, T., Villacis, J., Balasubramanian, J., E. Akman, Breg, F., Diwan, S. and Govindaraju, M. Component Architectures for Distributed Scientific Problem Solving. *IEEE Computational Science and Engineering*, 5 (2). 50-63. 1998.
14. Bray, T., Paoli, J. and Sperberg-McQueen, C.M. The Extensible Markup Language (XML) 1.0. 1998.
15. Brittenham, P. An Overview of the Web Services Inspection Language. 2001, www.ibm.com/developerworks/webservices/library/ws-wsiloer.
16. Casanova, H. and Dongarra, J. NetSolve: A Network Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11 (3). 212-223. 1997.
17. Casanova, H., Dongarra, J., Johnson, C. and Miller, M. Application-Specific Tools. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 159-180.
18. Catlett, C. In Search of Gigabit Applications. *IEEE Communications Magazine* (April). 42-51. 1992.
19. Catlett, C. and Smarr, L. Metacomputing. *Communications of the ACM*, 35 (6). 44--52. 1992.
20. Chamberlin, D. Xquery 1.0: An XML Query Language. W3C Working Draft 07, 2001.
21. Chin, R.S. and Chanson, S.T. Distributed Object-based Programming Systems. *ACM Computing Surveys*, 23 (1). 91-124. 1991.
22. Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. Web Services Description Language (WSDL) 1.1. W3C, Note 15, 2001, www.w3.org/TR/wsdl.
23. Clark, D.D., The Design Philosophy of the DARPA Internet Protocols. In *SIGCOMM Symposium on Communications Architectures and Protocols*, (1988), ACM Press, 106-114.
24. Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. In *10th IEEE International Symposium on High Performance Distributed Computing*, (2001), IEEE Press, 181-184.
25. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S. A Resource Management Architecture for Metacomputing Systems. In *4th Workshop on Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, 1998, 62-82.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

26. De Roure, D., Jennings, N. and Shadbolt, N. Research Agenda for the Semantic Grid: A Future e-Science Infrastructure. UK National eScience Center, 2002, www.semanticgrid.org.
27. Fallside, D.C. XML Schema Part 0: Primer. W3C, Recommendation, 2001, <http://www.w3.org/TR/xmlschema-0/>.
28. Foster, I. The Grid: A New Infrastructure for 21st Century Science. *Physics Today*, 55 (2). 42-47. 2002.
29. Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 259-278.
30. Foster, I. and Kesselman, C. (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
31. Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K. and Roy, A., A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proc. International Workshop on Quality of Service*, (1999), 27-36
32. Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S. Grid Services for Distributed Systems Integration. *IEEE Computer*, 35 (6). 2002.
33. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. A Security Architecture for Computational Grids. In *ACM Conference on Computers and Security*, 1998, 83-91.
34. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15 (3). 200-222. 2001. www.globus.org/research/papers/anatomy.pdf.
35. Fox, G., Balsoy, O., Pallickara, S., Uyar, A., Gannon, D. and Slominski, A. Community Grids. Community Grid Computing Laboratory, Indiana University, 2002.
36. Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S., Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *10th International Symposium on High Performance Distributed Computing*, (2001), IEEE Press, 55-66
37. Gannon, D., Bramley, R., Fox, G., Smullen, S., Rossi, A., Ananthakrishnan, R., Bertrand, F., Chiu, K., Farrellee, M., Govindaraju, M., Krishnan, S., Ramakrishnan, L., Simmhan, Y., Slominski, A., Ma, Y., Olariu, C. and Rey-Cenvaz, N., Programming the Grid: Distributed Software Components, P2P, and Grid Web Services for Scientific Applications. In *Grid 2001*, (2001)
38. Gasser, M. and McDermott, E., An Architecture for Practical Delegation in a Distributed System. In *Proc. 1990 IEEE Symposium on Research in Security and Privacy*, (1990), IEEE Press, 20-30
39. Getov, V., Laszewski, G.v., Philippsen, M. and Foster, I. Multiparadigm Communications in Java for Grid Computing. *Communications of the ACM*, 44 (10). 118-125. 2001.
40. Graham, S., Simeonov, S., Boubez, T., Daniels, G., Davis, D., Nakamura, Y. and Neyama, R. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Sams, 2001.
41. Grimshaw, A.S., Ferrari, A., Knabe, F.C. and Humphrey, M. Wide-Area Computing: Resource Sharing on a Large Scale. *IEEE Computer*, 32 (5). 29-37. 1999.
42. Grimshaw, A.S., Ferrari, A.J. and West, E.A. Mentat. In *Parallel Programming Using C++*, MIT Press, 1997, 383-427.
43. Grimshaw, A.S. and Wulf, W.A. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40 (1). 39-45. 1997.
44. Gullapalli, S., Czajkowski, K., Kesselman, C. and Fitzgerald, S. The Grid Notification Framework. Global Grid Forum, Draft GWD-GIS-019, 2001.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

45. Johnston, W. Realtime Widely Distributed Instrumentation Systems. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 75-103.
46. Johnston, W.E., Gannon, D. and Nitzberg, B., Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. In *Proc. 8th IEEE Symposium on High Performance Distributed Computing*, (1999), IEEE Press
47. Kreger, H. Web Services Conceptual Architecture. IBM Technical Report WCSA 1.0, 2001.
48. Liskov, B. Distributed Programming in Argus. *Communications of the ACM*, 31 (3). 300-312. 1988.
49. Litzkow, M. and Livny, M. Experience With The Condor Distributed Batch System. In *IEEE Workshop on Experimental Distributed Systems*, 1990.
50. Livny, M. High-Throughput Resource Management. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 311-337.
51. Mitchell, J.G., Gibbons, J., Hamilton, G., Kessler, P.B., Khalidi, Y.Y.A., Kougiouris, P., Madany, P., Nelson, M.N., Powell, M.L. and Radia, S.R., An Overview of the Spring System. In *COMPCON*, (1994), 122-131
52. Mockapetris, P.V. and Dunlap, K., Development of the Domain Name System. In *SIGCOMM*, (1988), ACM, 123-133
53. Mukhi, N. Web Service Invocation Sans SOAP. 2001, <http://www.ibm.com/developerworks/library/ws-wsif.html>.
54. Mullender, S. (ed.), *Distributed Systems*, 1989.
55. Nakada, H., Sato, M. and Sekiguchi, S. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems*, 15 (5-6). 649-658. 1999.
56. Nick, J.M., Moore, B.B., Chung, J.-Y. and Bowen, N.S. S/390 Cluster Technology: Parallel Sysplex. *IBM Systems Journal*, 36 (2). 172-201. 1997.
57. Oaks, S. and Wong, H. *Jini in a Nutshell*. O'Reilly, 2000.
58. Paton, N.W., Atkinson, M.P., Dialani, V., Pearson, D., Storey, T. and Watson, P. Database Access and Integration Services on the Grid. Manchester University, 2002.
59. Raman, S. and McCanne, S. A Model, Analysis, and Protocol Framework for Soft State-based Communication. *Computer Communication Review*, 29 (4). 1999.
60. Shapiro, M. SOS: An Object Oriented Operating System—Assessment and Perspectives. *Computing Systems*, 2 (4). 287-337. 1989.
61. Sharma, P., Estrin, D., Floyd, S. and Jacobson, V., Scalable Timers for Soft State Protocols. In *IEEE Infocom '97*, (1997), IEEE Press
62. Steen, M.v., Homburg, P., Doorn, L.v., Tanenbaum, A. and Jonge, W.d. Towards Object-based Wide Area Distributed Systems. In Carbrera, L.-F. and Theimer, M. eds. *International Workshop on Object Orientation in Operating Systems*, 1995, 224-227.
63. Steiner, J., Neuman, B.C. and Schiller, J., Kerberos: An Authentication System for Open Network Systems. In *Proc. Usenix Conference*, (1988), 191-202
64. Stevens, R., Woodward, P., DeFanti, T. and Catlett, C. From the I-WAY to the National Technology Grid. *Communications of the ACM*, 40 (11). 50-61. 1997.
65. Thomas, A. Enterprise Java Beans Technology: Server Component Model for the Java Platform. 1998, http://java.sun.com/products/ejb/white_paper.html.
66. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S. and Kesselman, C. Grid Services Specification. 2002, www.globus.org/research/papers/gsspec.pdf.
67. Tuecke, S., Engert, D., Foster, I., Thompson, M., Pearlman, L. and Kesselman, C. Internet X.509 Public Key Infrastructure Proxy Certificate Profile. IETF, Draft draft-ietf-pkix-proxy-01.txt, 2001.

This is a DRAFT document and a work in progress. Version: 6/22/2002.

Comments to foster@mcs.anl.gov, carl@isi.edu, jnick@us.ibm.com, tuecke@mcs.anl.gov

68. Villacis, J., M.Govindaraju, Stern, D., Whitaker, A., Breg, F., Deuskar, P., Temko, B., Gannon, D. and Bramley, R., CAT: A High Performance, Distributed Component Architecture Toolkit for the Grid. In *IEEE Intl Symp. on High Performance Distributed Computing*, (1999)
69. Zhang, L., Braden, B., Estrin, D., Herzog, S. and Jamin, S., RSVP: A new Resource ReSerVation Protocol. In *IEEE Network*, (1993), 8-18

Condor-G: A Computation Management Agent for Multi-Institutional Grids

James Frey, Todd Tannenbaum, Miron Livny
 Department of Computer Science
 University of Wisconsin
 Madison, WI 53706
 {jfrey, tannenba, miron}@cs.wisc.edu

Ian Foster, Steven Tuecke
 Mathematics and Computer Science Division
 Argonne National Laboratory
 Argonne, IL 60439
 {foster, tuecke}@mcs.anl.gov

Abstract

In recent years, there has been a dramatic increase in the amount of available computing and storage resources. Yet few have been able to exploit these resources in an aggregated form. We present the Condor-G system, which leverages software from Globus and Condor to allow users to harness multi-domain resources as if they all belong to one personal domain. We describe the structure of Condor-G and how it handles job management, resource selection, security, and fault tolerance.

1. Introduction

In recent years the scientific community has experienced a dramatic pluralization of computing and storage resources. The national high-end computing centers have been joined by an ever-increasing number of powerful regional and local computing environments. The aggregated capacity of these new computing resources is enormous. Yet, to date, few scientists and engineers have managed to exploit the aggregate power of this seemingly infinite Grid of resources. While in principle most users could access resources at multiple locations, in practice few reach beyond their home institution, whose resources are often far from sufficient for increasingly demanding computational tasks such as simulation, large scale optimization, Monte Carlo computing, image processing, and rendering. The problem is the significant "potential barrier" associated with the diverse mechanisms, policies, failure modes, performance uncertainties, etc., that inevitably arise when we cross the boundaries of administrative domains.

Overcoming this potential barrier requires new methods and mechanisms that meet the following three key user requirements for computing in a "Grid" that comprises resources at multiple locations:

- They want to be able to discover, acquire, and reliably manage computational resources

dynamically, in the course of their everyday activities.

- They do not want to be bothered with the location of these resources, the mechanisms that are required to use them, with keeping track of the status of computational tasks operating on these resources, or with reacting to failure.
- They do care about how long their tasks are likely to run and how much these tasks will cost.

In this article, we present an innovative distributed computing framework that addresses these three issues. The Condor-G system leverages the significant advances that have been achieved in recent years in two distinct areas: (1) security, resource discovery, and resource access in *multi-domain* environments, as supported within the Globus Toolkit [12], and (2) management of computation and harnessing of resources within a *single* administrative domain, specifically within the Condor system [20, 22]. In brief, we combine the inter-domain resource management protocols of the Globus Toolkit and the intra-domain resource management methods of Condor to *allow the user to harness multi-domain resources as if they all belong to one personal domain*. The user defines the tasks to be executed; Condor-G handles all aspects of discovering and acquiring appropriate resources, regardless of their location; initiating, monitoring, and managing execution on those resources; detecting and responding to failure; and notifying the user of termination. The result is a powerful tool for managing a variety of parallel computations in Grid environments.

Condor-G's utility has been demonstrated via record-setting computations. For example, in one recent computation a Condor-G agent managed a mix of desktop workstations, commodity clusters, and supercomputer processors at ten sites to solve a previously open problem in numerical optimization. In this computation, over 95,000 CPU hours were delivered over a period of less than seven days, with an average of 653 processors being active at any one time. In another case, resources at three

sites were used to simulate and reconstruct 50,000 high-energy physics events, consuming 1200 CPU hours in less than a day and a half.

In the rest of this article, we describe the specific problem we seek to solve with Condor-G, the Condor-G architecture, and the results obtained to date.

2. Large-scale sharing of computational resources

We consider a Grid environment in which an individual user may, in principle, have access to computational resources at many sites. Answering why the user has access to these resources is not our concern. It may be because the user is a member of some scientific collaboration, or because the resources in question belong to a colleague, or because the user has entered into some contractual relationship with a resource provider [14]. The point is that the user is authorized to use resources at those sites to perform a computation. The question that we address is *how to build and manage a multi-site computation* that uses those resources.

Performing a computation on resources that belong to different sites can be difficult in practice for the following reasons:

- Different sites may feature different authentication and authorization mechanisms, schedulers, hardware architectures, operating systems, file systems, etc.
- The user has little knowledge of the characteristics of resources at remote sites, and no easy means of obtaining this information.
- Due to the distributed nature of the multi-site computing environment, computers, networks, and subcomputations can fail in various ways.
- Keeping track of the status of different elements of a computation involves tedious bookkeeping, especially in the event of failure and dependencies among subcomputations.

Furthermore, the user is typically not in a position to require uniform software systems on the remote sites. For example, if all sites to which a user had access ran DCE and DFS, with appropriate cross-realm Kerberos authentication arrangements, the task of creating a multi-site computation would be significantly easier. But it is not practical in the general case to assume such uniformity.

The Condor-G system addresses these issues via a separation of concerns between the three problems of remote resource access, computation management, and remote execution environments:

- *Remote resource access* issues are addressed by requiring that remote resources speak standard

protocols for resource discovery and management. These protocols support secure discovery of remote resource configuration and state, and secure allocation of remote computational resources and management of computation on those resources. We use the protocols defined by the Globus Toolkit [12], a de facto standard for Grid computing.

- *Computation management* issues are addressed via the introduction of a robust, multi-functional *user computation management agent* responsible for resource discovery, job submission, job management, and error recovery. This Condor-G component is taken from the Condor system [20].
- *Remote execution environment* issues are addressed via the use of *mobile sandboxing* technology that allows a user to create a tailored execution environment on a remote node. This Condor-G component is also taken from the Condor system.

This separation of concerns between remote resource access and computation management has some significant benefits. First, it is significantly less demanding to require that a remote resource speak some simple protocols rather than to require it to support a more complex distributed computing environment. This is particularly important given that the deployment of production Grids [4, 18, 27] has made it increasingly common that remote resources speak these protocols. Second, as we explain below, careful design of remote access protocols can significantly simplify computation management.

3. Grid protocol overview

In this section, we briefly review the Grid protocols that we exploit in the Condor-G system: GRAM, GASS, MDS-2, and GSI. The Globus Toolkit provides open source implementations of each.

3.1. Grid security infrastructure

The Globus Toolkit's Grid Security Infrastructure (GSI) [13] provides essential building blocks for other Grid protocols and for Condor-G. This authentication and authorization system makes it possible to authenticate a user just once, using public key infrastructure (PKI) mechanisms to verify a user-supplied "Grid credential." GSI then handles the mapping of the Grid credential to the diverse local credentials and authentication/authorization mechanisms that apply at each site. Hence, users need not re-authenticate themselves each time they (or a program acting on their behalf, such as a Condor-G computation management service) access a new remote resource.

GSI's PKI mechanisms require access to a *private key* that they use to sign requests. While in principle a user's

private key could be cached for use by user programs, this approach exposes this critical resource to considerable risk. Instead, GSI employs the user's private key to create a *proxy credential*, which serves as a new private-public key pair that allows a proxy (such as the Condor-G agent) to make remote requests on behalf of the user. This proxy credential is analogous in many respects to a Kerberos ticket [26] or Andrew File System token.

3.2. GRAM protocol and implementation

The Grid Resource Allocation and Management (GRAM) protocol [10] supports remote submission of a computational request ("run program P") to a remote computational resource, and subsequent monitoring and control of the resulting computation. Three aspects of the protocol are particularly important for our purposes: security, two-phase commit, and fault tolerance. The latter two mechanisms were developed in collaboration with the UW team and are not yet part of the GRAM version included in the Globus Toolkit. They will be in the GRAM-2 protocol revision scheduled for later in 2001.

GSI security mechanisms are used in all operations to authenticate the requestor and for authorization. Authentication is performed using the supplied proxy credential, hence providing for single sign-on. Authorization implements local policy and may involve mapping the user's "Grid id" into a local subject name; however, this mapping is transparent to the user. Work in progress will also allow authorization decisions to be made on the basis of capabilities supplied with the request.

Two-phase commit is important as a means of achieving "exactly once" execution semantics. Each request from a client is accompanied by a unique sequence number, which is also included in the associated response. If no response is received after a certain amount of time, the client can repeat the request. The repeated sequence number allows the resource to distinguish between a lost request and a lost response. Once the client has received a response, it then sends a "commit" message to signal that job execution can commence.

Resource-side fault tolerance support addresses the fact that a single "resource" may often contain multiple processors (e.g., a cluster or Condor pool) with specialized "interface" machines running the GRAM server(s) that maintain the mapping from submitting client to local process. Consequently, failure of an interface machine may result in the remote client losing contact with what is otherwise a correctly queued or executing job. Hence, our GRAM implementation logs details of all active jobs to stable storage at the client side, allowing this information to be retrieved if a GRAM server crashes and is restarted. This information can include details of

how much standard output and error data has been received, thus permitting a client to request resending of this data after a crash of client or server.

3.3. MDS protocols and implementation

The Globus Toolkit's MDS-2 provides basic mechanisms for discovering and disseminating information about the structure and state of Grid resources [9]. The basic ideas are simple. A resource uses the Grid Resource Registration Protocol (GRRP) to notify other entities that it is part of the Grid. Those entities can then use the Grid Resource Information Protocol (GRIP) to obtain information about resource status. These two protocols allow us to construct a range of interesting structures, including various types of directories that support discovery of interesting resources. GSI authentication is used as a basis for access control.

3.4. GASS

The Globus Toolkit's Global Access to Secondary Storage (GASS) service [7] provides mechanisms for transferring data between a remote HTTP, FTP, or GASS server. In the current context, we use these mechanisms to stage executables and input files to a remote computer. As usual, GSI mechanisms are used for authentication.

4. Computation management: the Condor-G agent

Next, we describe the Condor-G computation management service (or Condor-G agent).

4.1. User interface

The Condor-G agent allows the user to treat the Grid as an entirely local resource, with an API and command line tools that allow the user to perform the following job management operations:

- submit jobs, indicating an executable name, input/output files and arguments;
- query a job's status, or cancel the job;
- be informed of job termination or problems, via callbacks or asynchronous mechanisms such as email;
- obtain access to detailed logs, providing a complete history of their jobs' execution.

There is nothing new or special about the semantics of these capabilities, as one of the main objectives of Condor-G is to preserve the look and feel of a local resource manager. The innovation in Condor-G is that these capabilities are provided by a personal desktop

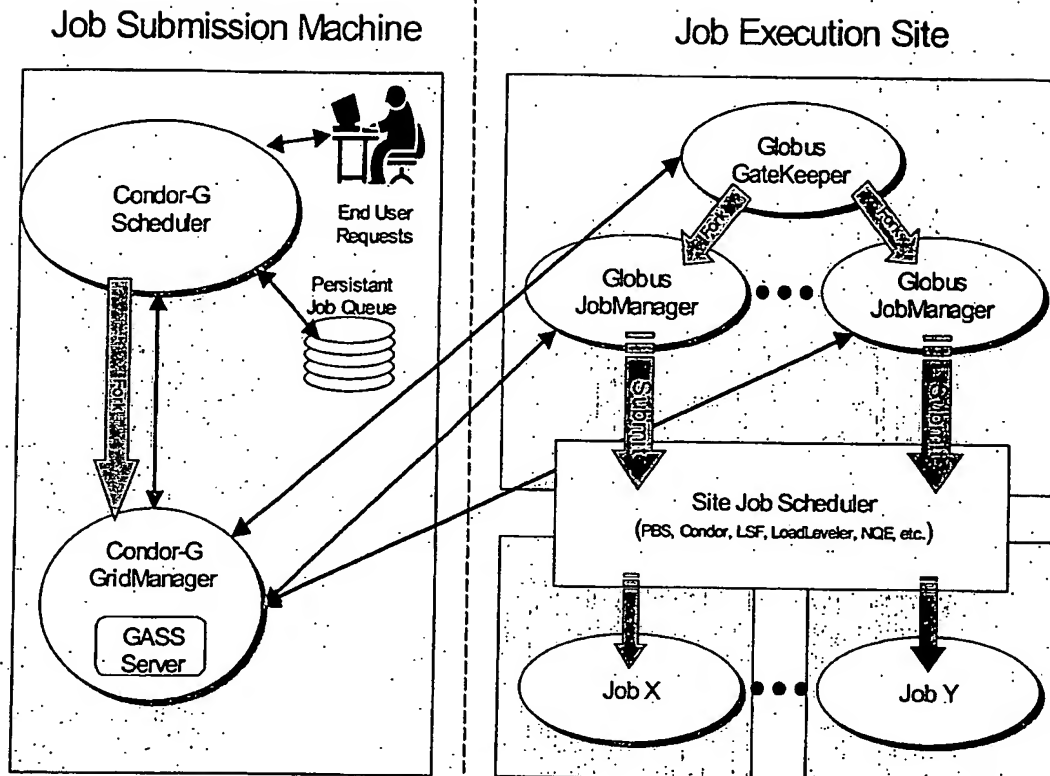


Figure 1. Remote execution by Condor-G on Globus-managed resources

agent and supported in a Grid environment, while guaranteeing fault tolerance and exactly-once execution semantics. By providing the user with a familiar and reliable single access point to all the resources he/she is authorized to use, Condor-G empowers end-users to improve the productivity of their computations by providing a unified view of dispersed resources.

4.2. Supporting remote execution

Behind the scenes, the Condor-G agent executes user computations on remote resources on the user's behalf. It does this by using the Grid protocols described above to interact with machines on the Grid and mechanisms provided by Condor to maintain a persistent view of the state of the computation. In particular, it:

- stages a job's standard I/O and executable using GASS,
- submits a job to a remote machine using the revised GRAM job request protocol, and
- subsequently monitors job status and recovers from remote failures using the revised GRAM protocol

and GRAM callbacks and status calls, while

- authenticating all requests via GSI mechanisms.

The Condor-G agent also handles resubmission of failed jobs, communications with the user concerning unusual and erroneous conditions (e.g., credential expiry, discussed below), and the recording of computation on stable storage to support restart in the event of its failure.

We have structured the Condor-G agent implementation as depicted in Figure 1. The Scheduler responds to a user request to submit jobs destined to run on Grid resources by creating a new *GridManager* daemon to submit and manage those jobs. One *GridManager* process handles all jobs for a single user and terminates once all jobs are complete. Each *GridManager* job submission request (via the modified two-phase commit GRAM protocol) results in the creation of one *Globus JobManager* daemon. This daemon connects to the *GridManager* using GASS in order to transfer the job's executable and standard input files, and subsequently to provide real-time streaming of standard output and error. Next, the *JobManager* submits the jobs to the execution site's local scheduling system. Updates

on job status are sent by the JobManager back to the GridManager, which then updates the Scheduler, where the job status is stored persistently as we describe below. When the job is started, a process environment variable points to a file containing the address/port (URL) of the listening GASS server in the GridManager process. If the address of the GASS server should change, perhaps because the submission machine was restarted, the GridManager requests the JobManager to update the file with the new address. This allows the job to continue file I/O after a crash recovery.

Condor-G is built to tolerate four types of failure: crash of the Globus JobManager, crash of the machine that manages the remote resource (the machine that hosts the GateKeeper and JobManager), crash of the machine on which the GridManager is executing (or crash of the the GridManager alone), and failures in the network connecting the two machines.

The GridManager detects remote failures by periodically probing the JobManagers of all the jobs it manages. If a JobManager fails to respond, the GridManager then probes the GateKeeper for that machine. If the GateKeeper responds, then the GridManager knows that the individual JobManager crashed. Otherwise, either the whole resource management machine crashed or there is a network failure (the GridManager cannot distinguish these two cases). If only the JobManager crashed, the GridManager attempts to start a new JobManager to resume watching the job. Otherwise, the GridManager waits until it can reestablish contact with the remote machine. When it does, it attempts to reconnect to the JobManager. This can fail for two reasons: the JobManager crashed (because the whole machine crashed), or the JobManager exited normally (because the job completed during a network failure). In either case, the GridManager starts a new JobManager, which will resume watching the job or tell the GridManager that the job has completed.

To protect against local failure, all relevant state for each submitted job is stored persistently in the scheduler's job queue. This persistent information allows the GridManager to recover from a local crash. When restarted, the GridManager reads the information and reconnects to any of the JobManagers that were running at the time of the crash. If a JobManager fails to respond, the GridManager starts a new JobManager to watch that job.

4.3. Credential management

A GSI proxy credential used by the Condor-G agent to authenticate with remote resources on the user's behalf is given a finite lifetime so as to limit the negative consequences of its capture by an adversary. A long-lived Condor-G computation must be able to deal with

credential expiration. The Condor-G agent addresses this requirement by periodically analyzing the credentials for all users with currently queued jobs. (GSI provides query functions that support this analysis.) If a user's credentials have expired or are about to expire, the agent places the job in a hold state in its queue and sends the user an e-mail message explaining that their job cannot run again until their credentials are refreshed by using a simple tool. Condor-G also allows credential alarms to be set. For instance, it can be configured to e-mail a reminder when less than a specified time remains before a credential expires.

Credentials may have been forwarded to a remote location, in which case the remote credentials need to be refreshed as well. At the start of a job, the Condor-G agent forwards the user's proxy certificate from the submission machine to the remote GRAM server. When an expired proxy is refreshed, Condor-G not only needs to refresh the certificate on the local (submit) side of the connection, but it also needs to re-forward the refreshed proxy to the remote GRAM server.

To reduce user hassle in dealing with expired credentials, Condor-G could be enhanced to work with a system like MyProxy [23]. MyProxy lets a user store a long-lived proxy credential (e.g. a week) on a secure server. Remote services acting on behalf of the user can then obtain short-lived proxies (e.g. 12 hours) from the server. Condor-G could use these short-lived proxies to authenticate with and forward to remote resources and refresh them automatically from the MyProxy server when they expire. This limits the exposure of the long-lived proxy (only the MyProxy server and Condor-G have access to it).

4.4. Resource discovery and scheduling

We have not yet addressed the critical question of how the Condor-G agent determines where to execute user jobs. A number of strategies are possible.

A simple approach, which we used in the initial Condor-G implementation, is to employ a user-supplied list of GRAM servers. This approach is a good starting point for further development.

A more sophisticated approach is to construct a personal *resource broker* that runs as part of the Condor-G agent and combines information about user authorization, application requirements and resource status (obtained from MDS) to build a list of candidate resources. These resources will be queried to determine their current status, and jobs will be submitted to appropriate resources depending on the results of these queries. Available resources can be ranked by user preferences such as allocation cost and expected start or completion time. One promising approach to constructing

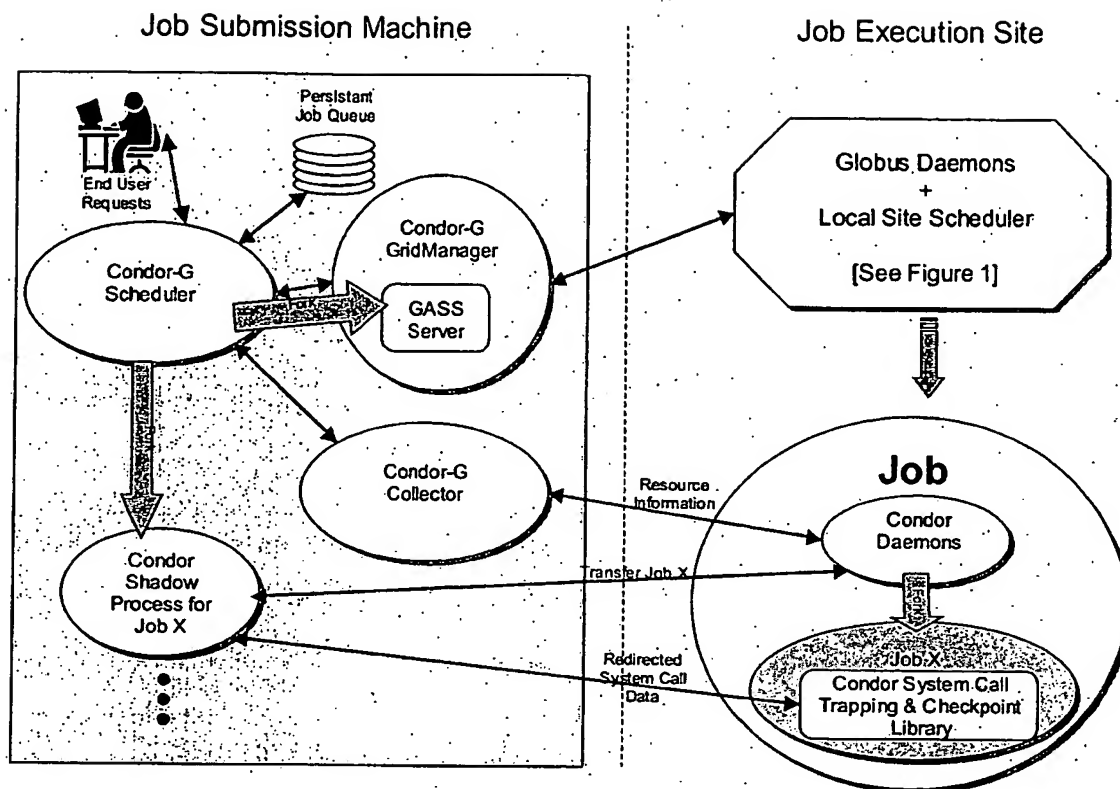


Figure 2. Remote job execution via GlideIn

such a resource broker is to use the Condor Matchmaking framework [25] to implement the brokering algorithm. Such an approach is described by Vazhkudai et al. [28]. They gather information from MDS servers about Grid storage resources, format that information and user storage requests into ClassAds, and then use the Matchmaker to make brokering decisions. A similar approach could be taken for computational resources for use with Condor-G.

In the case of high throughput computations, a simple but effective technique is to “flood” candidate resources with requests to execute jobs. These can be the actual jobs submitted by the user or Condor “GlideIns” as discussed below. Monitoring of actual queuing and execution times allows for the tuning of where to submit subsequent jobs and to migrate queued jobs.

5. GlideIn mechanism

The techniques described above allow a user to construct, submit, and monitor the execution of a task graph, with failures and credential expirations handled seamlessly and appropriately. The result is a powerful

management tool for Grid computations. However, we still have not addressed issues relating to what happens when a job executes on a remote platform where required files are not available and local policy may not permit access to local file systems. Local policy may also impose restrictions on the running time of the job, which may prove inadequate for the job to complete. These additional system and site policy heterogeneities can represent substantial barriers.

We address these concerns via what we call *mobile sandboxing*. In brief, we use the mechanisms described above to start on a remote computer not a user job, but a daemon process that performs the following functions:

- It uses standard Condor mechanisms to advertise its availability to a Condor Collector process, which is queried by the Scheduler to learn about available resources. Condor-G uses standard Condor mechanisms to match locally queued jobs with the resources advertised by these daemons and to remotely execute them on these resources [25].
- It runs each user task received in a “sandbox,” using system call trapping technologies provided by the Condor system [20] to redirect system calls

issued by the task back to the originating system. In the process, this both increases portability and protects the local system.

- It periodically checkpoints the job to another location (e.g., the originating location or a local checkpoint server) and migrates the job to another location if requested to do so (for example, when a resource is required for another purpose or the remote allocation expires) [21].

These various functions are precisely those provided by the daemon process that is run on any computer participating in a Condor pool. The difference is that in Condor-G, these daemon processes are started not by the user, but by using the GRAM remote job submission protocol. In effect, the Condor-G GlideIn mechanism uses Grid protocols to dynamically create a personal Condor pool out of Grid resources by "gliding-in" Condor daemons to the remote resource. Daemons shut down gracefully when their local allocation expires or when they do not receive any jobs to execute after a (configurable) amount of time, thus guarding against runaway daemons. Our implementation of this "GlideIn" capability submits an initial GlideIn executable (a portable shell script), which in turn uses GSI-authenticated GridFTP to retrieve the Condor executables from a central repository, hence avoiding a need for individual users to store binaries for all potential architectures on their local machines.

Another advantage of using GlideIns is that they allow the Condor-G agent to delay the binding of an application to a resource until the instant when the remote resource manager decides to allocate the resource(s) to the user. By doing so, the agent minimizes queuing delays by preventing a job from waiting at one remote resource while another resource capable of serving the job is available. By submitting GlideIns to *all* remote resources capable of serving a job, Condor-G can guarantee optimal queuing times to its users. One can view the GlideIn as an empty shell script submitted to a queuing system that can be populated once it is allocated the requested resources.

6. Experiences

Three very different examples illustrate the range and scale of application that we have already encountered for Condor-G technology.

An early version of Condor-G was used by a team of four mathematicians from Argonne National Laboratory, Northwestern University, and University of Iowa to harness the power of over 2,500 CPUs at 10 sites (eight Condor pools, one Cluster managed by PBS, and one supercomputer managed by LSF) to solve a very large optimization problem [3]. In less than a week the team logged over 95,000 CPU hours to solve more than 540 billion Linear Assignment Problems controlled by a

sophisticated branch and bound algorithm. This computation used an average of 653 CPUs during that week, with a maximum of 1007 in use at any one time. Each worker in this Master-Worker application was implemented as an independent Condor job that used Remote I/O services to communicate with the Master.

A group at Caltech that is part of the CMS Energy Physics collaboration has been using Condor-G to perform large-scale distributed simulation and reconstruction of high-energy physics events. A two-node Directed Acyclic Graph (DAG) of jobs submitted to a Condor-G agent at Caltech triggers 100 simulation jobs on the Condor pool at the University of Wisconsin. Each of these jobs generates 500 events. The execution of these jobs is also controlled by a DAG that makes sure that local disk buffers do not overflow and that all events produced are transferred via GridFTP to a data repository at NCSA. Once all simulation jobs terminate and all data is shipped to the repository, the Condor-G agent at Caltech submits a subsequent reconstruction job to the PBS system that manages the reconstruction cluster at NCSA.

Condor-G has also been used in the GridGaussian project at NCSA to prototype a portal for running Gaussian98 jobs on Grid resources. This Portal uses GlideIns to optimize access to remote resources and employs a shared Mass Storage System (MSS) to store input and output data. Users of the portal have two requirements for managing the output of their Gaussian jobs. First, the output should be reliably stored at MSS when the job completes. Second, the users should be able to view the output as it is produced. These requirements are addressed by a utility program called G-Cat that monitors the output file and sends updates to MSS as partial file chunks. G-Cat hides network performance variations from Gaussian by using local scratch storage as a buffer for Gaussian's output, rather than sending the output directly over the network. Users can view the output as it is received at MSS using a standard FTP client or by running a script that retrieves the file chunks from MSS and assembles them for viewing.

7. Related work

The management of batch jobs within a single distributed system or domain has been addressed by many research and commercial systems, notably Condor [20], DQS [17], LSF [29], LoadLeveler [16], and PBS [15]. Some of these systems were extended with restrictive and ad hoc capabilities for routing jobs submitted in one domain to a queue in a different domain. In all cases, both domains must run the same resource management software. With the exception of Condor, they all use a resource allocation framework that is based on a system-

wide collection of queues—each representing a different class of service.

Condor flocking [11] supports multi-domain computation management by using multiple Condor flocks to exchange load. The major difference between Condor flocking and Condor-G is that Condor-G allows inter-domain operation on remote resources that require authentication, and uses standard protocols that provide access to resources controlled by other resource management systems, rather than the special-purpose sharing mechanisms of Condor.

Recently, various research and commercial groups have developed software tools that support the harnessing of idle computers for specific computations, via the use of simple remote execution agents (workers) that, once installed on a computer, can download problems (or, in some cases, Java applications) from a central location and run them when local resources are available (i.e. SETI@home [19], Entropia, and Parabon). These tools assume a homogeneous environment where all resource management services are provided by their own system. Furthermore, a single master (i.e., a single submission point) controls the distribution of work amongst all available worker agents. Application-level scheduling techniques [5, 6] provide “personalized” policies for acquiring and managing collections of heterogeneous resources. These systems employ resource management services provided by batch systems to make the resources available to the application and to place elements of the application on these resources. An application-level scheduler for high-throughput scheduling that takes data locality information into account in interesting ways has been constructed [8]. Condor-G mechanisms complement this work by addressing issues of uniform remote access, failure, credential expiry, etc. Condor-G could potentially be used as a backend for an application-level scheduling system.

Nimrod [2] provides a user interface for describing “parameter sweep” problems, with the resulting independent jobs being submitted to a resource management system; Nimrod-G [1] generalizes Nimrod to use Globus mechanisms to support access to remote resources. Condor-G addresses issues of failure, credential expiry, and interjob dependencies that are not addressed by Nimrod or Nimrod-G.

8. Acknowledgment

This research was supported by the NASA Information Power Grid program.

9. References

- [1] Abramson, D., Giddy, J., and Kotler, L., “High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?”, *IPDPS2000*, IEEE Press, 2000.
- [2] Abramson, D., Sosc, R., Giddy, J., and Hall, B., “Nimrod: A Tool for Performing Parameterized Simulations Using Distributed Workstations”, *Proc. 4th IEEE Symp. on High Performance Distributed Computing*, 1995.
- [3] Anstreicher, K., Brixius, N., Goux, J.-P., and Linderoth, J., “Solving Large Quadratic Assignment Problems on Computational Grids”, *Mathematical Programming*, 2000.
- [4] Beiriger, J., Johnson, W., Bivens, H., Humphreys, S., and Rhea, R., “Constructing the ASCI Grid”, *Proc. 9th IEEE Symposium on High Performance Distributed Computing*, IEEE Press, 2000.
- [5] Berman, F., “High-Performance Schedulers”, Foster, I. and Kesselman, C. eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, pp. 279-309.
- [6] Berman, F., Wolski, R., Figueira, S., Schopf, J., and Shao, G., “Application-Level Scheduling on Distributed Heterogeneous Networks”, *Proc. Supercomputing '96*, 1996.
- [7] Bester, J., Foster, I., Kesselman, C., Tedesco, J., and Tuecke, S., “GASS: A Data Movement and Access Service for Wide Area Computing Systems”, *Sixth Workshop on I/O in Parallel and Distributed Systems*, May 5, 1999.
- [8] Casanova, H., Obertelli, G., Berman, F., and Wolski, R., “The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid”, *Proc. SC2000*, 2000.
- [9] Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C., “Grid Information Services for Distributed Resource Sharing”, 2001.
- [10] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S., “A Resource Management Architecture for Metacomputing Systems”, *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
- [11] Epema, D.H.J., Livny, M., Dantzig, R.v., Evers, X., and Pruyn, J., “A Worldwide Flock of Condors: Load Sharing among Workstation Clusters”, *Future Generation Computer Systems*, 12, 1996.
- [12] Foster, I. and Kesselman, C., “Globus: A Toolkit-Based Grid Architecture”, Foster, I. and Kesselman, C. eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, pp. 259-278.

- [13] Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S., "A Security Architecture for Computational Grids", *ACM Conference on Computers and Security*, 1998, pp. 83-91.
- [14] Foster, I., Kesselman, C., and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *Intl. J. Supercomputer Applications*, (to appear), 2001. <http://www.globus.org/research/papers/anatomy.pdf>.
- [15] Henderson, R. and Tweten, D., "Portable Batch System: External Reference Specification", 1996.
- [16] IBM, "Using and Administering IBM LoadLeveler, Release 3.0", IBM Corporation SC23-3989, 1996.
- [17] Institute, S.C.R., "DQS 3.1.3 User Guide", Florida State University, Tallahassee, 1996.
- [18] Johnston, W.E., Gannon, D., and Nitzberg, B., "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid", *Proc. 8th IEEE Symposium on High Performance Distributed Computing*, IEEE Press, 1999.
- [19] Korpela, E., Werthimer, D., Anderson, D., Cobb, J., and Lebofsky, M., "SETI@home: Massively Distributed Computing for SETI", *Computing in Science and Engineering*, 3(1), 2001.
- [20] Litzkow, M., Livny, M., and Mutka, M., "Condor - A Hunter of Idle Workstations", *Proc. 8th Intl Conf. on Distributed Computing Systems*, 1988, pp. 104-111.
- [21] Litzkow, M., Tannenbaum, T., Basney, J., and Livny, M., "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System", University of Wisconsin-Madison Computer Sciences, Technical Report 1346, 1997.
- [22] Livny, M., "High-Throughput Resource Management", Foster, I. and Kesselman, C. eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, pp. 311-337.
- [23] Novotny, J., Tuecke, S., and Welch, V., "An Online Credential Repository for the Grid: MyProxy", to appear in *HPDC10*.
- [24] Papakhian, M., "Comparing Job-Management Systems: The User's Perspective", *IEEE Computational Science & Engineering*, (April-June) 1998. See also <http://pbs.mrj.com>.
- [25] Raman, R., Livny, M., and Solomon, M., "Resource Management through Multilateral Matchmaking", *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, Pittsburgh, Pennsylvania, August 2000, pp. 290-291.
- [26] Steiner, J., Neuman, B.C., and Schiller, J., "Kerberos: An Authentication System for Open Network Systems", *Proc. Usenix Conference*, 1988, pp. 191-202.
- [27] Stevens, R., Woodward, P., DeFanti, T., and Catlett, C., "From the I-WAY to the National Technology Grid", *Communications of the ACM*, 40(11), 1997, pp. 50-61.
- [28] Vazhkudai, S., Tuecke, S., and Foster, I., "Replica Selection in the Globus Data Grid", *Proc. Of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*, IEEE Computer Society Press, May 2001, pp. 106-113.
- [29] Zhou, S., "LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems", *Proc. Workshop on Cluster Computing*, 1992.

Grid Computing Q&A with Benny Souder, Vice President, Distributed Database Development, Database and Application Server Technologies

General Q&As

Q: Please explain the basic concept of grid computing.

At the highest level, the central idea of Grid computing is computing as a utility. By that, we mean that you shouldn't care where your data resides, or what computer processes your request. You should be able to request information or computation and have it delivered – as much as you want, and whenever you want. This is analogous to the way electric utilities work, in that you don't know where the generator is, or how the electric grid is wired, you just ask for electricity, and you get it. The goal is to make computing a utility, a commodity, and ubiquitous. Hence the name, The Grid.

This view of utility computing is, of course, a "client side" view. From the "server side", or behind the scenes, the Grid is about resource allocation, information sharing, and high availability. Resource allocation to ensure that all those that need or request resources are getting what they need, that resources are not standing idle while requests are going unserved. Information sharing to make sure that the information users and applications need is available where and when it is needed. High availability since all the data and computation must always be there, just like a utility company must always provide electric power.

Q: How did the concept of grid computing begin? What industries primarily use it?

The idea of grid computing began in the academic and research communities. One of the earliest organizations active in Grids was CERN, which is also where the Web got its start. In some ways, the Web and the Grid have interesting parallels. Both are disruptive technologies, both began in research institutions, both spread to commercial enterprises. And, as they spread to commercial enterprises, both evolve in ways that are different than originally intended.

The first adopters of grid computing have been the financial, energy, and scientific industries. These are the commercial sectors that often adopt new technologies early. But more and more companies are starting to understand that grid computing has benefits for them, regardless of their industry. We're seeing more and more activity around the Grid in all industries, and we expect that to continue.

Q: How is Oracle involved in grid computing?

Oracle has been involved in grid computing for years, as both an end-user and a vendor. We think that makes us unique among the major software vendors.

As a Grid user, Oracle uses a grid to develop its database product. Oracle's grid enables us to build the database faster, and with higher quality. It allows us to allocate resources to specific development projects when we need to. It gives us much more computing power than any other alternative computing investment would give us. We

believe that using a Grid gives us competitive advantages in our industry: Quality, productivity, and time to market. And, our use of Grids gives us insight into the problems our users will face adopting Grid computing, and helps us understand how to make our customers successful with Grid computing.

As a Grid vendor, we think we can help customers gain the same kinds of benefits we've gained from Grid computing. Oracle has a strong technology stack, available today, which enterprises can leverage to reap grid computing benefits. Oracle has key technology differentiators that make Oracle offerings for Grid computing unique. We think that the Grid needs to be open, interoperable, and standards based. Therefore, we are working with the Global Grid Forum to help develop Grid standards. We're excited by Grid computing. We think Grid computing is the next big thing, and we think that it is already starting to happen. We think that as customers come to understand the Grid, and Oracle's capabilities, they'll become excited, too.

Q: What is the Global Grid Forum (GGF)?

The Global Grid Forum is a standards body that is developing standards for Grid computing. It is comprised of a set of committees and working groups that focus on various aspects of Grid computing. The committees and working groups are composed of participants from academic, research, and, increasingly, commercial companies. Oracle is working with GGF to help develop Grid standards.

A related organization is Globus. Globus is a project to develop open source Grid software. It predates GGF and unlike GGF it has full time staffing.

Q: Why does Oracle believe grid computing is the next big thing after the Internet?
The time is ripe for grid computing. There are a number of threads which, taken together, will make Grids unstoppable.

- In today's enterprises, people are concerned about affordability. Enterprises are looking at ways of reducing costs and increasing the efficiencies of their processes and systems. Grid computing offers exactly that. Grid computing increases the utilization of enterprise resources. Grid computing is a way to consolidate your hardware, eliminating islands of underutilized computers. Instead, you can create centralized pools of computing and allocate computing resources to the priorities of your organization.*
- In hardware, every vendor has announced or is delivering "blades". Computer blades offer the lowest cost computing power, sometimes as much as 80% less than SMP. These blades can easily be assembled into "blade farms", which are the most effective and scalable form of commodity computing. And, these blade farms are now being fitted with interconnects, making them hardware clusters. As such, they form the most cost effective form of commodity clusters, which we believe is the future architecture of computing.*

- *In software, Linux continues to grow faster than any other OS. Today, Linux cannot scale to large SMP. But since blades are 1-4 cpus, Linux runs well on them today. The economic advantage of blades over SMP will cause blades to dominate, and since Linux already works well for blades, this will accelerate Linux growth. Finally, Linux has a price advantage, which becomes more important as the number of blades grows, again accelerating Linux adoption. So commodity clusters naturally go well with Linux, the commodity OS.*
- *In both the software and hardware industries, one of the big buzzwords at the moment is "virtualization". But nothing is more "virtual" than a utility. A lot of vendors are trying to claim that their new strategy is "virtualization", or "utility computing" – which is exactly what Grid computing is all about. We think that these people will soon understand this and embrace the Grid.*
- *In the technology industry, grid momentum is building. Some major vendors such as Oracle are offering grid-enabling technology. Others such as IBM are planning to offer grid-enabling technology. The Grid standards body, GGF, is in place and has support of all major technology vendors.*
- *In IT organizations, Grid momentum is also building. Grid technologies promise increased utilization of existing hardware. Grids can let you allocate your resources to meet the needs of your business, instead of having islands of computing that are idle or overloaded. And, as existing hardware needs to be replaced, blades offer the lowest cost. The economics are so compelling that enterprises have already started leveraging blade servers for grid computing.*

In addition to these trends, there's another reason why we believe the Grid is the next big thing. If you look at the Web, it is really about presentation of information over the Internet or your intranet. We think after presentation, the next logical step is processing. Processing information over the Internet or your intranet is exactly what the Grid is all about. So one way to think of it is that the Grid is the next phase of the Internet, after the Web. In 1997 it was hard to see everything the Web would become, but you could tell it was going to be big. That's the state of the Grid today.

Q: How will customers benefit from using Oracle's grids?

Customers using Oracle's Grids will realize higher resource utilization and lower costs. They will also benefit from Oracle's superior operational characteristics – portability, availability, security and scalability. Oracle portability ensures you get the same operational benefits on all platforms including Linux and commodity clusters. Only Oracle can truly scale, provide high availability, and dynamically provision resources on low cost commodity clusters. Oracle makes the Grid unbreakable – you cannot break an Oracle grid and you cannot break into an Oracle grid.

Oracle also has key Grid technology differentiators – such as Oracle Real Application Clusters, Oracle Streams, and Oracle Transportable Tablespaces, among many others. Most importantly, Oracle has a proven record of providing software for leading

platforms and environments. Oracle Grid customers can feel confident their investment in Oracle technology will be leveraged as the Grid evolves.

Q: What type of Oracle customers would be interested in grid computing?

Everyone wants to save money, control resource allocation, improve utilization, and get faster, better results. So every Oracle customer should be interested in Grid computing.

The key to success in today's economic reality is affordability. Customers with multiple databases on dedicated hardware can use Grid computing to consolidate their hardware resources, and realize increased utilization and efficiencies. As customers replace their current hardware with newer, lower cost commodity hardware, Grid computing will let them capitalize on this newer hardware and gain even greater savings and efficiency. Every Oracle customer can benefit from Grid computing. We'll make it easy for our customers to move to the Grid. In the past, we've made it easy to adopt advances such as Unix, Java, and the Internet. We'll do it again for the Grid.

Q: What would be the long-term benefits of using grids?

Behind the scenes, on the server side, Grid computing changes the way enterprises look at resources. In place of buying resources for individual applications, enterprises will buy resources for all of an enterprise's needs and provision them to individual applications on demand or based on policy. This leads to efficient utilization of enterprises resources and also provides tremendous reduction in development, deployment, and management costs. It allows management to decide how to deploy the computing resources of the organization, and quickly change that deployment as the needs of the organization change.

For users, or clients, Grid computing means that you no longer have to understand how or where the computation is performed. You never have to know when you are allowed to do certain things, or ask certain questions. Imagine if you could only plug in your hair dryer after the big generator on the edge of town started up, or if you could only watch the television if you turned off the lights! When these kinds of things happen to our power grid, we think there is a big problem – it's national news – but today these kinds of considerations are normal for computing users. The Grid means to change that.

Q: Why has grid computing taken so long to catch on?

It hasn't, really. If you look at the history of the Web you'll see the Grid is probably catching on faster than the Web did. And, like the Web, certain key problems have to be solved and enabling technologies have to be developed before the idea can really take off. To realize the benefits of the Grid requires a sophisticated software infrastructure and universal connectivity. The Internet provides universal connectivity. Now we're solving the infrastructure problems.

Q: What does Oracle have to offer that will help promote or stimulate the growth of grid computing?

That's a good question, because we want to promote Grid computing. We think that it is a good idea, that it will help our customers, and that it is a great model for our products. We will promote and stimulate the growth of Grid computing in two ways.

First, we want to make it easy for Oracle customers to move to Grid computing. We do that by educating them on what the Grid is, how it can help them, and how to get there. And, we make it easy to move existing applications built with Oracle to Grid computing. We think this is very important to our customers, but also to the Grid. The Oracle installed base is very large, and moving the installed base to Grid computing would increase the number of Grid computing users many, many times over. Now, of course, the entire universe of Oracle-based systems won't immediately move to Grid computing. But, we can help start the process and we think that's good.

Second, Oracle can support Grid standards and make sure Oracle products have the capabilities needed for Grid computing. Oracle's involvement in the Grid helps drive solutions to data management and information sharing challenges – Oracle understands data management and information sharing as few other companies do. By making our products work well for Grid computing, we make it easy to be successful with this new model of computing. We're taking our leading software products, and tailoring them to Grid computing. That is not only an endorsement but also is a very large contribution to Grid computing.

Q: How much can customers expect to pay for Oracle grids?

The short answer is there are no additional costs to move to Oracle grids. The same Oracle products you use today support Grid computing. In fact, if you've deployed applications using Oracle, they are most likely immediately portable to Grid computing, with little or no change.

In the future, we will include additional Grid features in Oracle products that will be available as you upgrade to newer product versions. Our strategy will be to continue to offer integrated software, so you can install it and immediately gain benefits without integrating the pieces or buying services. Delivering an integrated enterprise software stack is our business model, not selling services and customizations, pieces and parts.

And, Oracle is making it's Grid SDK available now at no charge on the Oracle Technology Network, OTN. As future versions of the Globus Toolkit are built, we'll continue to support them and offer versions that are integrated with the Oracle stack.

Q: Why do you keep saying, "Oracle resonates with the Grid"?

Because I don't think we see the Grid as a fashion. I think if you look at all the ideas we've tried to communicate in the last decade, they all line up, they all make sense with Grid computing. We've talked about commodity computing, clusters and cluster databases for 10 years. We've talked about consolidation as long as I can remember. We're alone among the major software vendors in that we've stuck to our belief in outsourcing, which is exactly in line with utility computing. Our themes around the Internet, Unbreakable, and Linux are perfectly aligned with the Grid.

Honestly, I can't think of a company that has consistently been on message with the themes that today we call Grid computing, and been on that message for the last 10 years. So I don't think Oracle is going to approach the Grid as "this years' fashion". I think the Grid resonates deep with the core values and beliefs of Oracle.

Technology Q&As

Q: What is the difference between data grids and computing grids?

Compute grids and data grids are really the same thing. They utilize the same infrastructure. With a compute grid, the focus is on providing a computing resource. With data grids, the focus is on providing a data resource. In reality, very few problems are strictly compute or data. Most enterprises will require sharing both compute and data resources. We think that for commercial users, the distinctions are not important. In fact, we question how useful the distinction is to any Grid user.

That said, there are a lot of different implementations using the term Grid.. To make it easier to understand, we've developed a taxonomy of Grids...a classification scheme, if you will.

There are three phases, Scavenging, Sharing, and Dedicating. Each phase has hallmarks that make it pretty easy to take a given implementation and classify it. Each phase offers some benefits, but succeeding phases offer more benefits. Customers enter the grid at any of the three phases, but gravitate to later phases to gain increasing benefits. Today, we've already progressed through the maturation cycle to where most companies enter at Sharing or Dedicating. Within a year, I expect most customers to enter directly at Dedicating. Dedicating is where we aim with our technology, since it is where we most differentiate and where customers find most value. I'm happy to spend time explaining this in more detail – and perhaps we should bring this information out in more detail, it helps people understand why all these "different" things have the same name. And it may help move users along to Dedicating, which is good for them and for us. Oracle employees, at least, seem to really like the taxonomy. The only downside is you can only tell the average listener so much, and time we spend explaining taxonomies is time we can't spend explaining why we are better. But, education is good, and having this taxonomy may come in useful.

Q: How is information shared in Oracle Grids?

Information can be shared in a variety of ways. For information that is not frequently accessed, it may be most efficient to access it remotely as required. Oracle provides distributed SQL features that can transparently query or update data in other Oracle databases, making the data appear to be local. For data stored in non-Oracle databases, our distributed SQL features work in conjunction with our gateways to make the data appear to be local, and appear to be in the Oracle database. We offer Transparent Gateways for a number of database systems and a free generic gateway that gives access to any ODBC-compliant database.

It may be more efficient to move the data in bulk and access it locally, if the data is frequently accessed. Oracle Transportable Tablespaces allow Oracle data files to be unplugged from a database, moved or copied to another location, and then plugged into another database. Unplugging or plugging a data file involves only reading or loading a small amount of metadata. This makes Transportable Tablespaces a very fast mechanism for moving Oracle data. Transportable Tablespaces also supports simultaneous mounting of read-only tablespaces by two or more databases.

Some data that needs to be shared as it is created or changed, rather than occasionally shared in bulk. Oracle Streams can stream data between databases, nodes or blade farms in a Grid. It provides unified framework for information sharing, combining message queuing, replication, events, data warehouse loading, notifications and publish/subscribe into a single technology. Oracle Streams can keep two or more copies in sync as updates are applied. Streams automatically captures database changes, propagates the changes to subscribing nodes, applies changes, and detects and resolves any conflicts. Streams can also be used directly by applications as a messages queuing feature, enabling communications between applications in the Grid.

Q: Could there be a “gridnet” – like an Internet?

Perhaps. If this does come to pass, it likely will be as the “Internet” and “intranets” are used today. So, there would be some support for free public utility computing over the Internet, but commercial users will likely keep most of their Grid inside the firewall, as companies do today with intranets.

Also, it is possible that using technology like VPN, service providers will offer outsourcing or hosting services across the Internet using Grid technologies. The same advantages Grid computing gives companies will be equally important to outsourcing suppliers – greater utilization will result in lower costs, which translate into lower rates for users, which could help to speed adoption of outsourcing. And, that’s as it should be, since outsourcing is very much computing as a utility. You don’t know where the computer is, and you don’t care, you get the applications, information, and computing you need.

Q: What are the benefits of using grids vs. using supercomputers?

Grids and supercomputers are not exclusive. A supercomputer can be a resource on a Grid. In fact, the quest for maximum utilization of expensive supercomputers was an early motivation for the Grid. Grid technology unlocks the potential of alternatives to supercomputers such as farms of inexpensive commodity server blades. These will provide much higher compute capacities at the fraction of the cost of supercomputers. The trick is getting software that can efficiently utilize a blade farm. For databases, Oracle is clearly the superior choice for blade farms.

Q: Are grids a good step or even a likely one for mainstream decision support?

Some of the earliest uses of Grid computing were to analyze massive amounts of scientific data. So, from the start, Grid computing has had a goal of supporting analytic operations and large amounts of data. Grid technologies enable efficient resource

utilization, and flexible resource allocation. That's just as good for DSS as it is for OLTP. Grid technology can make compute rich environments available to DSS applications, if that is your priority. But, perhaps more importantly, the right Grid technology can allow you to provision compute resources to DSS or OLTP as the needs of your business change.

For example, suppose you are a large internet retailer. At Christmas, you want to allocate all your resource to your website, to maximize sales and minimize response times. But, after Christmas, your website is virtually idle. Everyone bought everything at Christmas. Now, you have mountains of purchase and clickstream data. You want to allocate all your resources to analyze this data. That way, you can improve your marketing. You can get better plans for next year. If you have two isolated SMPs, one for the website and one for the data warehouse, this reallocation of resources is difficult or impossible. But, using Grid technology, you can reallocate. And with Oracle, you can reallocate easily.

Q: What does Oracle provide for its developers in the area of grid computing? Are there toolkits available?

Well, we're making the first version of the Oracle Grid SDK available on OTN. The SDK makes it easy to use the Oracle along with the Globus Toolkit. The Globus Toolkit is a set of useful components that can be used either independently or together to develop Grid applications. The first version of our SDK implements an open source, or reference version, of the APIs of the Globus Toolkit mapped to Oracle APIs. So, again, we've done the integration for you. Here's the details:

- Globus Resource Allocation Manager (GRAM): GRAM provides resource allocation and process creation, monitoring, and management services. GRAM implementations map requests expressed in a Resource Specification Language (RSL) into commands understood by local schedulers and computers. We've mapped GRAM to Oracle. This enables jobs specified in Globus RSL to be mapped to Oracle scheduled jobs, and to Oracle stored procedures. Thus, if you use GRAM, we've already done the work to integrate that with Oracle.*
- Monitoring and Discovery Service (MDS): MDS is an extensible Grid information service that combines data discovery mechanisms with the Lightweight Directory Access Protocol (LDAP). MDS provides a uniform framework for providing and accessing system configuration and status information such as compute server configuration, network status, or the locations of replicated data. MDS provides a Grid Resource Information Service (GRIS) that can be used to get details of an individual resource on the Grid. We've integrated with GRIS, to expose Oracle database attributes and properties through LDAP. This lets you find Oracle databases on a Grid, and determine if that particular database has the information or content you need.*

Q: What specific Oracle products are used to support grids?

It's important to understand the simplicity of the Oracle Grid strategy. Oracle is not building new products for Grid computing. Oracle is incorporating Grid capabilities

into all of its present products. So, the Oracle products that support Grid computing now, and in the future, are the same Oracle products you know today: Oracle 9i, Oracle iAS, and the technology stack built on top of them. We're offering support for Globus protocols and Globus resource discovery in our Grid SDK, which is free. When you move to Grid computing, you won't have to learn a new enterprise software stack, it will be the same stack you know how to use now.

Our goal is to allow everything built on the Oracle stack to move to Grid computing. We may not achieve this goal, but I think we will be able to allow the vast majority of Oracle-based applications to function in Grids. So, in a sense, every technology Oracle offers is relevant to Grids.

So, if you are interested in getting started with Grid computing, we have the technology you need. But, if you want to wait, we'll make it easy for you to adopt Grid computing when you are ready, and protect your investment in Oracle.

Q: Will Oracle incorporate its Web Services technology into its grid computing products?

Web services technology will be useful for Grid computing. Web services offer a well-defined communication mechanism. We have a complete Web Services technology offering, and it easily supports Grid computing. But, it is important to realize that web services are just one of the interfaces with which Grid entities can communicate. Because Grid users come from diverse backgrounds and are familiar with diverse development environments, Oracle will provide a choice of development environment and communication models.

Q: Will Oracle incorporate its clustering technology into its grid computing products?

Yes, RAC will be a key differentiator in the Grid. It enables the use of lowest cost hardware. It lets databases dynamically add and release resources. This ability to add and drop resources is critical to improving utilization and efficiency, and thus to reducing costs and improving productivity. It makes databases on this lowest cost hardware highly available. And, it lets you run real applications on this hardware.

Grid benefits



Grid computing goes far beyond sheer computing power. Today's operating environments must be resilient, flexible and integrated as never before. Organizations around the world are experiencing substantial benefits by implementing grids in critical business processes to achieve both business and technology benefits.

Business benefits

Accelerate time to results:

- can help improve productivity and collaboration
- can help solve problems that were previously unsolvable

Enable collaboration and promote operational flexibility:

- bring together not only IT resources but also people
- allow widely dispersed departments and businesses to create virtual organizations to share data and resources

Efficiently scale to meet variable business demands:

- create flexible, resilient operational infrastructures
- address rapid fluctuations in customer demands|needs
- instantaneously access compute and data resources to "sense and respond" to needs

Increase productivity:

- can help give end users uninhibited access to the computing, data and storage resources they need (when they need them)
- can help equip employees to move easily through product design phases, research projects and more — faster than ever

Leverage existing capital investments:

- can help you improve optimal utilization of computing capabilities
- can help you avoid common pitfalls of over-provisioning and incurring excess costs
- can free IT organizations from the burden of administering disparate, non-integrated systems

Technology benefits

Infrastructure optimization:

- consolidate workload management
- provide capacity for high-demand applications

- reduce cycle times

Increase access to data and collaboration:

- federate data and distribute it globally
- support large multi-disciplinary collaboration
- enable collaboration across organizations and among businesses

Resilient, highly available infrastructure:

- balance workloads
- foster business community
- enable recovery and failure

An Enabling Framework for Master-Worker Applications on the Computational Grid *

Jean-Pierre Goux[†]

Sanjeev Kulkarni[†]

Jeff Linderot[§]

Michael Yoder[†]

March 10, 2000

Abstract

We describe MW – a software framework that allows users to quickly and easily parallelize scientific computations using the master-worker paradigm on the computational grid. MW provides both a “top level” interface to application software and a “bottom level” interface to existing grid computing toolkits. Both interfaces are briefly described. We conclude with a case study, where the necessary Grid services are provided by the Condor high-throughput computing system, and the MW-enabled application code is used to solve a combinatorial optimization problem of unprecedented complexity.

*This work was supported in part by Grants No. CDA-9726385 and CDA-9623632 from the National Science Foundation.

[†]Department of Electrical and Computer Engineering, Northwestern University, and Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439, goux@mcs.anl.gov

[‡]Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton Street, Madison, WI 53706, {sanjeevk,yoderme}@cs.wisc.edu

[§]Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439, linderot@mcs.anl.gov

1 Introduction

By its very definition, the Grid [11] is a powerful and complex computing environment. In order to help harness its power, a large number of different programming efforts are underway that seek to provide robust middleware services [10] [14] [17] [9] [3] [21]. For users hoping to parallelize a large, single, coordinated application over the Grid, the overhead required to learn and assemble these Grid-enabling software components could (at this stage of their implementation) be discouraging. Thus, to enable a larger community of users to build applications running in parallel on the Grid, higher-level programming frameworks leveraging existing Grid services software are needed. NetSolve [4] provides an API to access and schedule Grid resources in a seamless way but it is not suited for writing non-embarrassingly parallel codes. Everyware [22] is a heroic effort that shows that an application can draw computational power transparently from the Grid, but Everyware is not abstracted as a programming tool at this stage of its implementation. CARMI/Wodi [20] was a useful programming interface for developing master-worker based parallel applications to run on the Grid, but it was strongly tied to the Condor-PVM [19] software tool, limited to applications with fixed work cycles, and finally abandoned.

Our abstract programming framework MW is a complete, easy to use tool whereby users can distribute large, diverse, scientific computations in a Grid computing environment. The focus is on parallel applications with weak synchronization and reasonably large grain size that can be fit into a master-worker paradigm without significant loss of efficiency. To parallelize such algorithms on Grid computing platforms, users must address issues such as fault tolerance, task scheduling, and interprocess communication. By handling some of these issues automatically and exposing others, MW provides an API for rapidly implementing Grid-enabled master-worker algorithms. MW also abstracts an Infrastructure Programming Interface (IPI) such that it can be ported to use various Grid software toolkits without any changes from the application developer. MW has been used in the MetaNEOS project [18] to implement efficient parallel numerical optimization algorithms with complex control structures. The marriage of efficient algorithms with Grid computational resources has allowed the solution of problems of record breaking sizes [2] [15].

The paper is organized as follows. In Section 2, we introduce MW, and we describe the interfaces to both application software and Grid infrastructure

software. Section 3 discusses additional features of MW that help developers build efficient and robust applications. Section 4 presents a case study where the Grid services are provided by Condor[17], and the application code is used to solve a combinatorial optimization problem of unprecedented complexity. Conclusions about this line of research are also given.

2 MW

MW is a software framework that allows a user to easily parallelize a master-worker application on Grid resources. MW is a set of C++ abstract classes providing interfaces to both application programmer and Grid-infrastructure programmer. To Grid-enable an application with MW, the application programmer must re-implement a small number of virtual functions. Likewise, to port the MW framework to a new Grid software toolkit, the Grid-infrastructure programmer need only re-implement a small number of virtual functions.

2.1 Infrastructure Interface

To distribute a master-worker computation on the Grid, we at least require software that can perform

- Communication – Portions of the computation and results must be passed between master and workers,
- Resource Management – The state of the available computational resources on the Grid must be known.

Our usage of the term *resource management* is a bit broader than most. In this context, resource management encompasses

- Resource request and detection – Asking for and identifying available processors,
- Infrastructure querying – Determining information about processors and the interconnections between them,
- Fault-detection – Noticing when processors leave the computation

- Remote execution – Starting processes on remote machines when they become available.

There are a number of tools being built that provide these basic services, as well as features necessary to other Grid applications (such as security and remote data access). The *Infrastructure programming interface* (IPI) abstracts the core communication and resource management requirements for master-worker applications into the MWRMComm class. To allow MW applications to interact with existing Grid-services software, a concrete instance of the abstract MWRMComm class is derived, where the functionality required by MWRMComm is provided by the services in the specific Grid software toolkit.

2.1.1 Communication

The sole communications functionality required by MWRMComm is that point-to-point messages can be sent between the master and the worker processes. As such, MWRMComm has the (pure) virtual functions:

- `pack(<type> array, int size)`
- `unpack(<type> array, int size)`
- `send(int to_whom, int message_tag)`
- `recv(int from_whom, int message_tag)`

All messages must be buffered by the MWRMComm implementation, and the `send()` function should be implemented as a nonblocking call. These design criteria are due to the fact that processors may disappear during the course of the computation. Since the Grid is heterogeneous, the `pack()` and `unpack()` functions must account for different native data types. In MWRMComm, the `recv()` routine should be implemented as a blocking function call.

2.1.2 Resource management

The application programmer may make a resource requests by calling the function `MWRMComm::set_target_num_workers(int num_workers)`. It is up to the MWRMComm implementation to make appropriate resource

requests in an attempt to garner this number of workers for the master-worker application, and also to make new requests if participating workers leave the computation.

An important design decision for MW is that both communication and resource management functionality is included in a common class. The reason behind this decision is that MW requires that all information about the state of the computational resources be passed to it in the form of messages with specific tags such as `HOSTADD` and `HOSTDELETE`. Thus, an implementation of the (blocking) `MWRMComm::recv()` function on the master process should not only test for incoming messages from workers, but also check for changes to the state of the existing computational resources and report these changes as messages.

When a `HOSTADD` message is received, the `MWRMComm` specification requires that the function call `MWRMComm::start_worker(MWWorkerID *w)` will (attempt to) start a remote process on the machine that has been added, and will assign a unique process identifier in the `MWWorkerID`. When a `HOSTDELETE` message is received, `MWRMComm` requires that the unique process identifier be packed in the message buffer.

A final important function in the `MWRMComm` class is `MWRMComm::get_worker_info(MWWorkerID *w)`. This function uses underlying Grid services to populate the `MWWorkerID` class with “useful” information about the remote processor. Data members of the `MWWorkerID` class include the architecture, operating system, amount of memory, disk space, and speed of the remote machine.

Clearly, this is not the entire specification of the `MWRMComm` class. Indeed, we consider the IPI that we have laid out in MW to be a work in progress. The interface will likely change, and additional functionality will be added as warranted. Due the layered design of MW, application programs will be shielded from the interface changes.

2.1.3 Example MWRMComm Implementations

There are currently two implementations of the `MWRMComm` class. Both rely on the resource management facilities provided by the Condor high-throughput computing system [17]. As such, the `MWDriver` must deal with many processor faults, since the default Condor behavior is to vacate a running process when the “owner” of the machine returns.

In one implementation, communication is done with PVM, and in the

other, communication is done by using Condor's remote I/O mechanism [16] to write a series of shared files. Preliminary plans are being made for a port to the Globus software toolkit [10]. Table 1 highlights how the Grid service software provides (or could provide) the functionality required by MWRMComm.

There are advantages and disadvantages to having MW act as an "upper middleware" layer between application code and Grid service software code. The additional software layer acts as a filter, hiding complexity of Grid service software, but also potentially hiding underlying functionality and knowledge of how the communication and resource management services are performed. A significant challenge is how to impart this functionality and knowledge to the application programmer, while still presenting a simple interface. MW errs on the side of simplicity, with the thought that additional Grid service functionality will be made available to the application programmer as needed.

An advantage of the layered approach is that some advances in Grid services software can be leveraged by the application programmers to increase application performance. For our Condor-based MWRMComm implementations, two examples include flocking [8], where geographically distributed Condor pools are conceptually linked as one, and glide-in [7], where processors from an existing Globus resource can be added to a Condor pool on a temporary basis. These advanced Condor features are used by the application presented in Section 4.

2.2 MW Application Programming Interface

In a companion work [13], we argue that many scientific applications can be parallelized quite effectively for a Grid environment by using the master-worker paradigm. Our specific experience is with algorithms for solving numerical optimization problems and many of these algorithms share the following characteristics:

- **Incremental Data Requirement** – A potentially large amount of data must be passed to worker processes at initialization, but thereafter, messages are small "incremental" changes to the initial data.
- **Weak Synchronization** – The ability to execute a task does not depend on the completion of a large number of other tasks.

Services	Condor-PVM	Condor-Files	Globus
Communi- cation	Messages buffered and passed through PVM <code>pvm_pk()</code> in XDR format.	Messages passed through shared worker files via Condor Remote I/O.	Messages passed and handled via Nexus <code>nexus_send_rsr()</code> .
Resource Request and Detection	Requests formulated with Condor Class Ads, served by Condor matchmaking, and detection is notified by <code>pvm_notify</code> .	Requests formulated with Condor Class Ads, served by Condor matchmaking and detected, by checking Condor logs.	Requests in Globus RSL handled and queued by GRAM via <code>gram_client_job_request()</code> .
Info Querying	Information collected via <code>condor_status</code> command	Information collected via <code>condor_status</code> command	Information queried from MDS via LDAP protocol.
Fault Detection	Faults detected by Condor-PVM and passed through <code>pvm_notify()</code> .	Faults detected by checking Condor logs.	Faults detected by HBM local monitors are collected by HBM data collector agent running on master.
Remote Execution	Job started by <code>pvm_spawn()</code> .	Job started by <code>condor_startd</code> daemon on remote resource	Job started by GRAM when requests are served

Table 1: Summary of How Grid Services are Provided

- **Dynamic Grain Size** – The computation can naturally be broken into portions of work of variable size.

The MW API was designed to provide an interface that would be *easy* for application programmers to use; but also would allow these algorithmic characteristics to be exploited to build efficient master-worker applications.

In order to parallelize an application with MW, the application programmer must re-implement three abstract base classes – MWDriver, MWTask, and MWWorker.

2.2.1 MWDriver

To create the MWDriver, the user need only implement four pure virtual functions:

- `get_userinfo(int argc, char *argv[])` – Processes arguments and does basic setup.
- `setup_initial_tasks(int *n, MWTask ***tasks)` – Returns a set of tasks for the computation to begin work on.
- `pack_worker_init_data()` – Packs the initial data to be sent to the worker upon startup. Use of this function allows the application to exploit an *incremental data requirement*.
- `act_on_completed_task(MWTask *task)` – Is called every time a task finishes. Some actions that the user could take include adding more tasks or making calculations based on the result of the task.

By carefully deciding on actions to take in the `act_on_completed_task()` method, the user can take advantage of a *weak synchronization* inherent in the parallel application.

The MWDriver manages a set of MWTasks and a set of MWWorkers to execute those tasks. The MWDriver base class handles workers joining and leaving the computation, assigns tasks to appropriate workers, and rematches running tasks when workers are lost. All this complexity is hidden from the application programmer. Further, the MWDriver offers more advanced functionality, as explained in Section 3.

2.2.2 MWTask

The MWTask is the abstraction of one unit of work. The class holds both the data describing that task and the results computed by the worker. By deciding on the size of the task, the application can use *dynamic grain size* to its advantage, easing contention at the master process, and increasing parallel efficiency. The derived task class must implement functions for sending and receiving its data between the master and worker. The names of these functions are self-explanatory: `pack_work()`, `unpack_work()`, `pack_results()`, and `unpack_results()`. These functions will call associated `pack()` and `unpack()` functions in the MWRMComm class:

2.2.3 MWWorker

The MWWorker class is the core of the worker executable. Two pure virtual functions must be implemented:

- `unpack_init_data()`– Unpacks the initialization information passed in the MWDriver's `pack_worker_init_data()`.
- `execute_task(MWTask *task)`– Given a task, computes the results.

After doing some basic initialization, the MWWorker sits in a simple loop. Given a task, it computes the results, reports the results back, and waits for another task. The loop finishes when the master asks the worker to end. It is an easy matter to bring in other libraries, such as highly optimized FORTRAN routines to the worker. They can be linked with the C++ code, and called by the `execute_task()` function.

3 Additional Functionality

In addition to the necessary services provided by the MWDriver and the MWRMComm implementation, users of the MW-framework benefit from a number of other useful features that are available through methods in the base MWDriver class.

3.1 Checkpointing

Because the MWDriver reschedules tasks when the processors running these tasks fail, applications running on top of MW are fault tolerant in the presence of all processor failures—except for the master processor. In order to make computations fully reliable, MWDriver offers features to logically checkpoint the state of the computation on the master process on a user-defined frequency. To enable checkpointing, the user must implement functions for writing and reading the state contained in its application's master and task classes. Use of the master checkpoint facility is demonstrated in Section 4.

3.2 Normalized Performance Measurement

The heterogeneous and dynamic nature of the Grid makes application performance difficult to assess. Standard performance measures such as wall clock time and cumulative CPU time do not separate application code and computing platform performance. By normalizing the CPU time spent on a given task with the performance of the corresponding worker, the MWDriver aggregates time statistics that are comparable between runs. The normalization factor can be based on vendor information such as MIPS or KFLOPS, if this information is available from the underlying Grid service software. Alternatively, MW allows the user to register an application specific benchmark task that is sent to all workers that join the computational pool. The speed at which the benchmark task is completed is used as the normalization factor.

If we make the following definitions:

- $\alpha(i)$ – Worker i performance normalization factor,
- $U(i)$ – Worker i uptime,
- $w(j)$ – Index of worker who solved task j ,
- $t(j)$ – User time spent by worker $w(j)$ at solving j ,
- \mathcal{W} – Wall clock time,
- T – Cumulative workers CPU time.

We can then define the following statistics:

- \mathcal{T} – Normalized cumulative time :

$$\mathcal{T} = \sum_{j \in \mathcal{J}} \alpha(w(j)) * t(j)$$

- \mathcal{P} – Equivalent Pool Performance :

$$\mathcal{P} = \frac{\sum_{i \in \mathcal{I}} \alpha(i) * U(i)}{\sum_{i \in \mathcal{I}} U(i)}$$

- \mathcal{N} – Average number of workers :

$$\mathcal{N} = \frac{\sum_{i \in \mathcal{I}} U(i)}{\mathcal{W}}$$

- η – Parallel efficiency :

$$\eta = \frac{\sum_{j \in \mathcal{J}} t(j)}{\sum_{i \in \mathcal{I}} U(i)}$$

Table 3.2 shows the variations of performance statistics between runs of a Grid-enabled application (presented in Section 4). The same problem instance was solved eight times, each time on a different set of processors. A user-defined benchmark task was used to define the normalization factor.

	Mean	Std. Dev.	Min	Max
\mathcal{W}	915	1019	489	1780
\mathcal{T}	22182	27900	8844	37671
\mathcal{P}	5864	341	5739	6054
\mathcal{N}	7.27	7.16	3.2	12.4
η	27.5	21.7	16	39
η	0.87	0.07	0.84	0.92

Table 2: Mean, Variance and Extreme Value on 8 different runs.

As expected, the statistics show the large variance of \mathcal{W} and \mathcal{T} . However, there is little variance of \mathcal{P} , which can therefore be used to do comparisons between runs and assess the application performance.

3.3 Task Scheduling

Internally, the MWDriver manages a list of workers and a list of tasks. Task scheduling is accomplished by assigning the first task in the task list to the first idle worker in the worker list. In MWDriver, there is an interface to specify that the task list be ordered by a user-defined key, ensuring that “important” tasks are performed first. The worker list may be similarly ordered, so that “good” machines are the first to receive tasks. By default, the worker list is ordered using the machine KFLOPS information (if provided by the Grid software implementing MWRmComm), or by the benchmark factor if the user has registered an application specific benchmark task.

While this is a rudimentary scheduling algorithm, it has proven sufficient for all applications implemented to date with MW. The applications have had no need to match specific tasks with specific workers. Also the applications are not data-intensive, so use of advanced services such as the Network Weather Service ?? to improve scheduling has not been warranted.

4 Application to Combinatorial Optimization

MW has been used in the MetaNEOS project [18] to implement several grid enabled parallel optimization solvers [6] [12] [15]. One solver has been specialized to solve the quadratic assignment problem (QAP) [5]. Despite its simple statement—to minimize the assignment cost of n facilities to n locations—it is extremely difficult to solve even modest sized instances of the QAP. Problems with $n > 20$ are difficult; problems with $n > 30$ have not even been attempted yet. By embedding a new relaxation technique [1] into a branch-and-bound framework, and implementing the resulting solver within MW, we managed to solve what is regarded by experts in the field as the most difficult QAP instance to provable optimality [2].

In order to use the computational resources with maximal efficiency, the parallelization strategy of the branch-and-bound tree search has been carefully designed. Issues such as the proper ordering of the task list and the selection of the grain size were carefully considered in order to minimize communication overhead and contention at the master process without introducing large parallel search anomalies. By using the intuitive MW API, implementing the parallel version of the sequential branch-and-bound code was extremely simple and fast.

The MW-ized QAP application code was compiled to use the Condor/File-Based MWRMComm implementation. Our computational pool was composed of

- University of Wisconsin Condor pool workstations,
- a University of Wisconsin dedicated Linux cluster,
- a University of New Mexico “flocked” Condor pool,
- the National Institute for Nuclear Physics, Bologna, Italy, “flocked” Condor pool, and
- the Argonne National Laboratory SGI/Origin2000 acquired via Globus through the glide-in mechanism.

The pool is depicted in Figure 1.

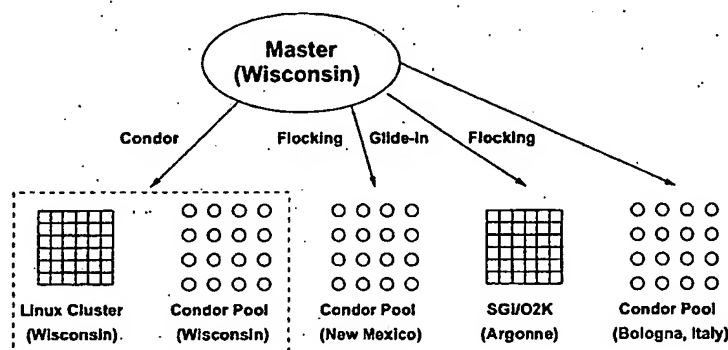


Figure 1: The Computational Pool.

Figure 2 depicts the evolution of the number of machines of each type during our run. At 11:30AM a glide-in request was made for 32 SGI processors on Argonne’s O2K for a period of 12 hours. At 6:30 PM, the Condor scheduling daemon was reconfigured to allow flocking with the INFN Condor pool in Bologna, Italy. The job was stopped manually at 11PM, and we restarted it at 8AM from the master’s checkpoint file, as explained in Section 3.1.

Over the course of the computation, an average of 211.3 machines and with a peak of 285. The parallel efficiency obtained during the run was $\eta =$

0.83. The average performance of the computational pool was 195 times the performance of one of the dedicated Linux nodes. Neglecting parallel search anomalies, the solution of this problem in sequential would have required around over 177 days of computation with the sequential algorithm on a dedicated Linux node. The marriage of Grid resources with the advanced algorithm allowed the solution of a heretofore unsolved problem.

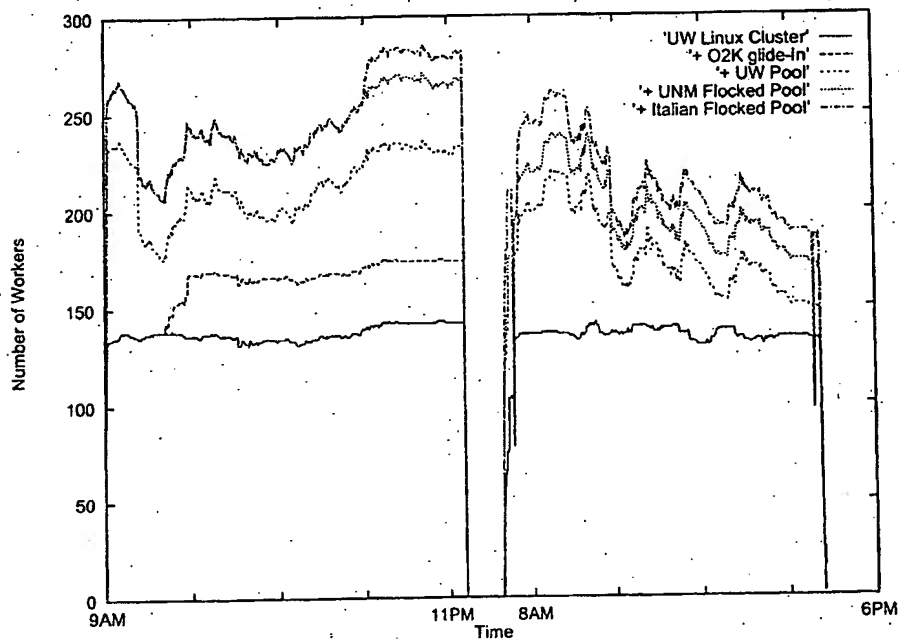


Figure 2: Number of Workers

5 Conclusions and Future Work

MW has allowed algorithm developers to bring together a large number of heterogeneous, geographically dispersed resources to solve extremely large problems. The simple API of MW provided a convenient programming model enabling the user to focus on algorithmic features without worrying on the details of setting up computations, and the IPI has allowed a better portability of the resulting code to different grid computing environments.

It is the continued goal of this work to draw further application developers by providing a simple interface, access to Grid resources, and useful functionality at no expense to the application code. We also wish to entice Grid infrastructure developers to support MW by providing a simple, well-defined interface, and interesting and useful applications. There is still work to be done to turn these goals into realities.

Further Information about MW is available from

<http://www.cs.wisc.edu/condor/mw>

Acknowledgments

The authors would like to sincerely thank the whole Condor team for their tireless efforts, and their timely responses to our never-ending queries. In particular, we thank Jaime Frey for his help with the glide-in feature, and Miron Livny for his enthusiastic support. We also thank Nate Brixius and Kurt Anstreicher for allowing us to use their application code in the case study, and for keeping us excited about building a parallel Grid interface.

References

- [1] K. Anstreicher and N. Brixius. A new bound for the quadratic assignment problem based on convex quadratic programming. Technical report, Department of Management Sciences, University of Iowa, 1999. Available from <http://www.biz.uiowa.edu/faculty/anstreicher/qapqp.ps>.
- [2] K. Anstreicher, N. Brixius, J.-P. Goux, G. Hudek-Davis, and J. Linderoth. Location theory gives rise to QAP problem. *data link*, March 2000. The Alliance Online Technical News Letter, available from <http://www.ncsa.uiuc.edu/SCD/Alliance/datalink/0003/QA.Condor.html>.
- [3] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff. Charlotte: Meta-computing on the Web. *International Journal on Future Generation Computer Systems*, 15:559-570, 1999.

- [4] Henri Casanova and Jack Dongarra. NetSolve: Network enabled solvers. *IEEE Computational Science and Engineering*, 5(3):57-67, 1998.
- [5] E. Cela. *The Quadratic Assignment Problem - Theory and Algorithms*. Kluwer, 1998.
- [6] Q. Chen, M. Ferris, and J. Linderoth. Fatcop 2.0: Advanced features in an opportunistic mixed integer programming solver. Data Mining Institute 99-11, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1999. Available from <http://www.mcs.anl.gov/metaneos/fatcop2.ps>.
- [7] The Condor Team. *Extending your Condor pool by Gliding into Globus-controlled machines*, 2000. <http://www.cs.wisc.edu/condor/manual/v6.1/2.12Extending-your.html>.
- [8] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Journal on Future Generation Computer Systems*, 12:67-85, 1996.
- [9] G. Fagg, K. Moore, and J. Dongarra. Scalable networked information processing environment (SNIPE). *International Journal on Future Generation Computer Systems*, 15:595-605, 1999.
- [10] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. J. Supercomputer Applications*, 1997. Available as <ftp://ftp.globus.org/pub/globus/papers/globus.ps.gz>.
- [11] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.
- [12] J.-P. Goux and Sven Leyffer. Mixed-integer nonlinear programming on metacomputing platform. Working Paper, 1999.
- [13] J.-P. Goux, J. Linderoth, and M. Yoder. Metacomputing and the master-worker paradigm. Preprint ANL/MCS-P792-0200, Mathematics and Computer Science Division, Argonne National Laboratory, 2000.
- [14] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Legion: An operating system for wide-area computing. Available as <http://legion.virginia.edu/papers/CS-99-12.ps.Z>, 1999.

- [15] J. Linderoth and S. Wright. A cutting plane code for stochastic programming on metacomputers. Invited Presentation at the INFORMS National Meeting, November 1999.
- [16] M. Litzkow. Remote Unix – Turning idle workstations into cycle servers. In *Proceedings of Usenix Summer Conference*, 1987.
- [17] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP*, 11, 1997. Available from http://www.cs.wisc.edu/condor/doc/htc_mech.ps.
- [18] The MetaNEOS Project. *Metacomputing Environments for Optimization*, 2000. <http://www.mcs.anl.gov/metaneos>.
- [19] J. Pruyne and M. Livny. Providing resource management services to parallel applications. In *Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing*, 1994. Available as http://www.cs.wisc.edu/condor/doc/condor_pvm_framework.ps.Z.
- [20] J. Pruyne and M. Livny. Interfacing Condor and PVM to harness the cycles of workstation clusters. *Journal on Future Generations of Computer Systems*, 12:53–65, 1996.
- [21] M. van Steen, P. Homburg, and A. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency*, 7:70–78, 1999.
- [22] R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su. Running Everywhere on the computational Grid. In *SC99 Conference on High-performance Computing*, 1999. Available from <http://www.cs.utk.edu/~rich/papers/ev-sc99.ps.gz>.

Communication Pattern Based Node Selection for Shared Networks

Srikanth Goteti
ComStock, Division of Interactive Data Corp
600 Mamaroneck Avenue
Harrison, NY, 10528
srikanth.goteti@cmstk.com

Jaspal Subhlok
University of Houston
Department of Computer Science
Houston, TX 77204
jaspal@uh.edu

Abstract

Selection of the most suitable nodes on a network to execute a parallel application requires matching the network status to the application requirements. We propose and validate a novel two step approach that exploits the knowledge of the communication structure of the application to address this problem. In the first step, a small set of candidate node groups are selected as potential sites of application execution, by analyzing the network status information and the communication patterns used by the application. The second step is based on the concept of a communication skeleton, which is a short running program that generates the dominant communication operations of the application it represents. The communication skeleton is executed on all candidate groups of nodes. The node group selected for application execution is the one that achieves the best performance on the communication skeleton. This approach leads to customized node selection and is particularly well suited to situations where available network information is of poor quality or expected communication performance cannot be modeled accurately. We motivate this approach, describe a prototype implementation, and present performance results for NAS Parallel Benchmarks executing on a shared network testbed.

1. Introduction

Selection of computation nodes to execute a parallel application is a central problem for computing on shared clusters and computation grids [7, 8]. Node selection based on CPU considerations has been addressed by several systems, some well known examples being Condor [9] and LSF [20]. The problem of node selection is significantly more complex when the communication needs of applications must also be taken into account. The main reasons for the additional complexity are that the communication properties cannot be associated with individual nodes, network sta-

tus changes dynamically, and the availability of network resources is difficult to measure and predict accurately. The state of the art in resource selection with communication considerations can be paraphrased as follows. The status of the network is measured and predicted with tools such as Network Weather Service [19] and Remos [10, 11], and this information is analyzed to identify a good group of available computation nodes and network paths for application execution. Some research projects have focused on getting the best general group of execution nodes [3, 14, 18] while others have developed procedures customized for a particular application or application class [4, 5, 6, 12].

This approach to node selection has the fundamental drawback that the decisions made are, at best, only as good as the accuracy with which the network status was measured and future network performance predicted. The performance that a network delivers to an application can vary significantly from the performance predicted by network measurement tools for a variety of reasons, some of which are as follows:

- Network status and prediction information may be outdated. Measurement of network properties, such as available bandwidth, can be intrusive and expensive, and the cost rises rapidly with the size of the network. Hence, it may be practical only to perform measurements relatively infrequently while the network state changes continuously.
- Network tools typically measure the unused network capacity or the network bandwidth achieved by a specific measurement probe. However, the relationship between available network capacity and the performance achieved by communication operations in an application is complex, and depends on other factors also, such as the network transport protocols in use. For example, the bandwidth that a TCP stream achieves on a busy network route partly depends on the number of other TCP streams using the links in the route.

- The performance of collective communication operations, common in parallel applications, is very difficult to estimate on a shared cluster or a grid environment. We are not aware of any tools that have proven their effectiveness in this respect. In particular, interference between multiple application communication streams sharing the same network path, is very difficult to model.

The point is that the expected performance of an application's communication operations inferred from network measurement tools can be significantly different from the actual performance for several different reasons. This limits the effectiveness of any node selection procedure entirely based on network measurements.

This research pursues a new approach to node selection motivated by the above discussion. The centerpiece of our methodology is the concept of a *performance skeleton* of an application, which is defined as a synthetically generated short running program that has the same fundamental execution characteristics as the application it represents, but with no semantic relevance. The execution time of the performance skeleton program on a given set of nodes reflects the execution time of the application under the same conditions, but possibly scaled down by multiple orders of magnitude. In this approach, the performance of the performance skeleton on a group of nodes determines the likelihood of those nodes being chosen for execution of the corresponding application since the performance of the skeleton is closely related to the performance of the application. This methodology eliminates the impact of inherent inaccuracy in network measurement and modeling. A performance skeleton is constructed ahead of time and executed prior to application execution to drive node selection.

This paper addresses only a part of the challenge of employing a performance skeleton based approach to node selection. The results presented are restricted to sharing of communication resources only. We assume that all available computation nodes have the same available computation capacity but communication properties of the network links connecting the nodes are varying. Hence, node selection is based on bandwidth considerations only. In this scenario, a performance skeleton needs to be faithful to the original application in terms of communication behavior only. Hence, in order to be more accurate, we will refer to them as *communication skeletons* in this paper.

A communication skeleton is a short running program, and that is the key to keeping the run-time overhead of this approach acceptable. However, the number of possible groups of nodes that are candidates for application execution grows combinatorially with the total number of available nodes. Hence, it is not practical to execute even a short running communication skeleton on every candidate group of nodes. Therefore, we employ a separate procedure to se-

lect a set of candidate node groups from all available nodes. This algorithm is based on the information about the network status obtained from network measurement tools and the information about the communication pattern of an application, which is computed in a preprocessing phase. The final group of execution nodes is selected based on the execution time of the performance skeleton program on the candidate node groups.

This paper is organized as follows. The node selection framework is described in section 2. Section 3 describes our prototype implementation and results from experiments to validate the node selection procedure. Section 4 explains the capabilities and limitations of our approach and implementation, and discusses ongoing and future work. Section 5 contains conclusions.

2 Node selection framework

We first outline the main steps and components of the node selection framework.

The first two steps are performed ahead of time, once for each application.

1. Identify the main communication patterns of the candidate application.
2. Construct the communication skeleton of the application.

The following subsequent steps are performed at the time the application has to be scheduled for execution.

3. Obtain current network status information.
4. Identify a small set of candidate node groups for execution by employing a node selection algorithm based on the network status and application's communication pattern.
5. Execute the communication skeleton program on each candidate group of nodes. Select the node group with the lowest execution time to schedule the application.

We now discuss each of the above steps in more detail.

2.1 Identification of communication pattern

The node selection framework relies heavily on the knowledge of the communication patterns in the application that has to be executed. These are captured by executing the application in a preprocessing phase on a controlled testbed and monitoring the message traffic between nodes. The procedure has been discussed in detail in [13, 15] in a

related context. The methodology completely relies on system monitoring on the testbed while the application is executing and does not require application knowledge or access to the source code. The communication structure of NAS benchmark programs inferred from such runtime measurements is illustrated in Figure 2.

2.2 Communication skeleton program

The communication skeleton of an application is a synthetically generated program that replicates the dominant communication patterns employed by the application. As stated earlier, the size and pattern of the messages exchanged by the nodes executing an application are inferred by monitoring execution on a controlled testbed. Automatic construction of skeletons from this information is an important long term goal of our research. However, for the results presented in this paper, manually constructed communication skeletons were employed. A program that performs a set of representative message exchanges along the communication routes used by the application qualifies as a communication skeleton of the application.

2.3 Network status information

Our network status measurement module employs Network Weather Service [19], a freely available distributed resource monitoring system. NWS gathers system level resource information, such as CPU load and available bandwidth, for network connected compute nodes. We employ NWS to measure the available bandwidth between all nodes that can be used to execute an application. This step yields a graph with compute nodes as graph nodes and available bandwidth between them as graph edges. We will refer to this graph as the network status graph.

2.4 Node selection algorithm

The first step in the process of node selection is a procedure that analyzes the network status graph to choose a set of "good" candidate groups of nodes for application execution. Another input to the node selection procedure is the application structure, basically the number of nodes required to execute the application and pairs of nodes that communicate in the main data exchange patterns. The objective of this algorithm is to determine the group of nodes for which the minimum of the available bandwidth between communicating nodes is maximized. The reason for choosing this particular criterion is that the time for completion of a collective communication step in parallel programs is typically determined by the lowest bandwidth communication path rather than the average available bandwidth on communication paths.

This communication pattern based algorithm for node selection is presented in Figure 1. The algorithm is similar to the one that was introduced by Subhlok et. al. in [14], but with one important difference. The algorithm in Figure 1 attempts to optimize performance over network paths that are utilized by an application, while the algorithm in [14] treated all network paths connecting executing nodes equally. For example, in the algorithm in Figure 1, if the main application communication pattern is an all to all data exchange between computing nodes, then the network path between each pair of nodes is optimized. However, if the main communication pattern takes the form of a one dimensional ring, then only the paths composing the ring are considered for optimization.

We informally explain the node selection algorithm stated in Figure 1. Suppose the goal of the algorithm is to select m optimal nodes. The algorithm starts with the network status graph and repeatedly removes the edge with the minimum available bandwidth from the graph. At every step, the algorithm verifies that there are m nodes that are connected in a way that satisfies the communication pattern of the application. (e.g., if the communication pattern is a ring, there must be a path from one node to another such that a ring can be completed.) A path from one node to another can include network routers but not other computation nodes. When removing the minimum available bandwidth edge leads to a situation where m such nodes cannot be found, then the algorithm stops. The last step is reversed and a set of m nodes is selected.

The algorithm as presented in Figure 1 selects a single group of optimal nodes, but our framework is based on selection of a set of candidate node groups. In practice, the algorithm is easily modified for usage in our framework by backtracking the last few edge deletions and selecting all feasible node groups at that point.

2.5 Final node selection with communication skeletons

For final node selection, the communication skeleton program is executed on each group of candidate nodes selected by the node selection algorithm described above. The group of nodes on which the communication skeleton achieves the best performance is selected for application execution. An important consideration in this step is to not execute the communication skeleton concurrently on intersecting groups of nodes since execution on one group of nodes is likely to impact performance on other groups. Note that the communication skeleton program is short running, typically a few seconds long, and hence this stage is not likely to make a significant impact on the turnaround time of an application.

Input: A connected network status graph G . An application pattern graph A with m compute nodes representing the number nodes needed by the application and the application communicating pattern. That is, there is an edge between a pair of graph nodes in A if the link between the corresponding application nodes is included in the main application communication pattern. Assume that the number of computed nodes in G is at least m .

Output: A graph M containing m nodes that represents a mapping of the application to the compute nodes that maximizes the minimum bandwidth between any pair of communicating nodes as represented in A .

1. $M = \text{null}$
2. Attempt to find a subgraph $\text{new}M$ of G such that there is a path between any two nodes of $\text{new}M$ if there is an edge between the corresponding nodes of A . If no such graph exists, set $\text{new}M = \text{null}$.
3. If ($\text{new}M == \text{null}$)
 return (M)
 Else
 $M = \text{new}M$
4. Remove the edge with the minimum available bandwidth from G .
 Goto Step 2.

Figure 1. Algorithm to select a set of nodes in a network status graph in order to maximize the minimum available bandwidth between any pair of communicating nodes based on a given application communication pattern graph.

3 Experiments and results

A prototype of the node selection framework discussed in this paper was implemented and validated on a network testbed. We first describe the experiments performed and then discuss the results.

3.1 Experimental setup

The testbed for the experiments is a compute cluster composed of 10 Intel Xeon dual CPU 1.7 GHz machines connected by 100Mbps Ethernet links and a full crossbar switch. All experimental results are based on the MPI implementation of the NAS Parallel Benchmarks [2, 16]. The codes used are BT (Block Tridiagonal solver), CG (Conjugate Gradient), IS (Integer Sort), LU (LU Solver), MG (Multigrad) and EP (Embarrassingly parallel). All programs are compiled using GNU g77, (Fortran) compiler except IS, which is compiled with the gcc (C) compiler. The MPICH implementation of MPI is used. The bandwidth between computation nodes was managed with the Linux advanced networking *iproute2* [1] in order to sim-

ulate limited bandwidth availability due to competing network traffic. *iproute2* works by intercepting the network packets and passing them through artificial queues to simulate bandwidth limitations.

3.2 Building communication patterns and communication skeletons

In order to make an application “ready” for automatic node selection, the main communication patterns have to be discovered and a communication skeleton program has to be created in a preprocessing phase. For the NAS benchmark programs included in this study, the basic communication patterns were derived by execution on a dedicated testbed with system level monitoring of network traffic. We will skip the details of these measurements but they are discussed in [15, 17]. The results are illustrated in Figure 2.

The next objective is to construct the communication skeletons. The NAS benchmarks are available in several sizes labeled Class S, W, A, B, and C, in increasing order of the size of data structures and execution time. Class S benchmarks run within a few seconds on a small cluster,

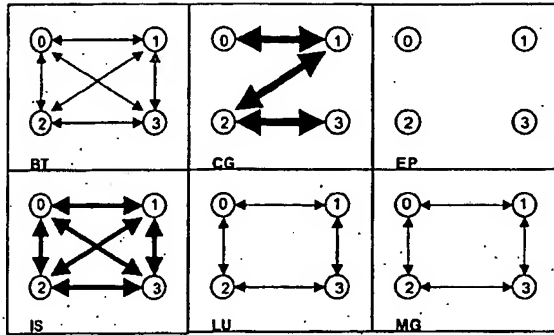


Figure 2. Dominant communication patterns during execution of NAS benchmarks. The thickness of the lines reflects the generated communication bandwidth.

while Class C benchmarks require a fairly large system to run at an acceptable speed. We chose class A benchmarks as the target applications to optimize. We also chose the corresponding class S benchmarks as the communication skeletons for the class A benchmarks, since they closely resemble each other and are likely to be very good skeleton programs. Our longer term goal in this research is to automatically construct performance skeletons. So, clearly, we are “cheating” by simply using a good skeleton program that happens to be available in this case. The reason is that we did not want the results to be impacted by the quality of the skeletons that we constructed since automatically building good skeletons is an open research problem that is not the focus of this paper. Hence, the results we obtained could be labeled as optimistic. However, based on other ongoing research, it is our firm belief that, in the near future, it will be possible to automatically generate skeletons of the quality that we have used for our experiments.

3.3 Automatic node selection

In order to evaluate node selection in the presence of network traffic, experiments were performed with varying available bandwidth caused by simulated network traffic. The available bandwidth on the network links connecting the computation nodes was controlled in the following manner. At any given time, every network link was assumed to be shared by a varying number of other traffic streams. If S streams are already sharing a network link, the bandwidth available to our application with fair sharing is assumed to be $1/(S + 1)$. Every 30 seconds, one traffic stream would randomly enter or leave the system, with a resultant increase or decrease in the available bandwidth on the affected link. The bandwidth, however, was never allowed to go below

10Mbps and cannot exceed the link capacity of 100Mbps. Based on the above simulation model, the actual bandwidth was controlled with the *iproute2* toolset.

Each NAS benchmark program was executed repeatedly on 4 nodes selected by our prototype node selection module based on the framework presented. NWS was employed to measure the available bandwidth between pairs of compute nodes on the network and build a network status graph. The node selection algorithm presented in Figure 1 was used to select the best three groups of nodes every time a benchmark program had to be scheduled and executed. Subsequently, the corresponding communication skeleton was executed on each of the three groups of nodes, and the group on which it performed the best was the selected node group. The benchmark program was then executed on those nodes and the execution time was measured and compared to the execution time on a dedicated testbed. For comparison, the procedure was also performed with two other node selection methods. The three node selection procedures that were evaluated and compared against each other are as follows:

1. **Pattern based:** The framework presented in this paper.
2. **All-all:** The nodes were selected using the network information, on the basis of maximizing the minimum available bandwidth between any pair of selected nodes. This approach requires a detailed analysis of the network status graph, but does not use any application specific information such as the communication pattern, and does not employ communication skeletons.
3. **Random:** Nodes were selected at random for reference.

The performance achieved by the benchmark programs on nodes selected by each of these methods was measured. The experiments were repeated a large number of times to get statistically meaningful results. For each benchmark program, the average execution time with each node selection procedure was computed and compared to the execution time of the same benchmark on a dedicated testbed with full bandwidth available on all links. The average slowdown due to link sharing for each benchmark program and each node selection procedure is presented in Figure 3.

3.4 Results

We observe from Figure 3 that each benchmark program performs significantly better when pattern based node selection is employed as compared to random node selection. On average, the percentage slowdown with random node selection is 40%, while that with pattern based node selection

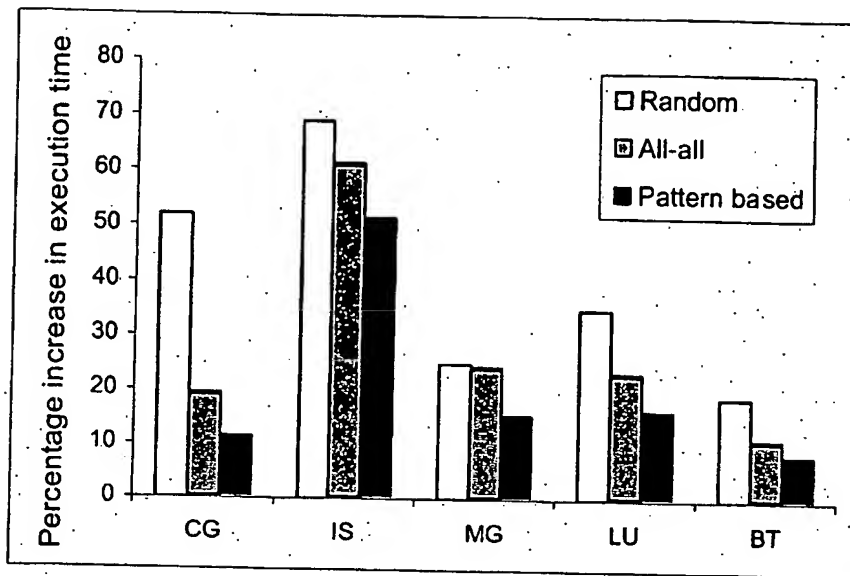


Figure 3. Percentage increase in the execution time of the NAS benchmarks due to network sharing for different node selection methods.

is around 20.2%. Hence, under these particular simulated conditions, the slowdown due to competing network traffic is reduced by half with good, application specific, node selection.

The node selection procedure labeled "all-all" can be considered a state of the art approach to node selection, but one that does not use the new concepts introduced in this paper. In the all-all method, the basis for node selection is maximizing the minimum available bandwidth between every pair of selected nodes, as described in [14]. The method entails a detailed analysis of the network status graph, but no consideration is given to the application communication structure, and communication skeletons are not used. The average slowdown with all-all node selection is 27.6% versus 20.2% for the pattern based framework. Hence, in our experiments, the pattern based approach to node selection reduces the slowdown due to link sharing by roughly a quarter as compared to a good approach that does not consider application communication patterns.

We observe that the general percentage slowdown as well as the relative performance with different node selection procedures varies dramatically across the programs in the NAS benchmark suite. EP benchmark is not included in the graph as it has no communication, and hence its performance is unaffected by the changes in available bandwidth and does not depend on the node selection procedure employed. The CG and IS benchmarks show the greatest percentage increase in execution time with random node selec-

tion. We observe from Figure 2 that these benchmarks are the most bandwidth hungry of the suite, which is the reason they are most affected by a reduction in the available bandwidth.

The pattern based scheme performs better than the random and all-all schemes for every application but there are significant differences. The maximum improvement in performance is observed for the CG benchmark. We speculate that the reason is that only 3 pairs of nodes communicate in CG as shown in Figure 2. A smart node selection procedure has a better chance of finding a relatively small number of "good" network paths as compared to finding good paths between every pair of selected nodes. This translates to finding 3 good paths versus 6 good paths for 4 nodes. Further, as mentioned earlier, CG is among the most communication intensive programs in the suite, and therefore, its performance is most sensitive to network path selection.

Another observation is that the relative improvement with pattern based node selection, as compared to all-all node selection, is lowest for IS and BT benchmarks. This is not surprising since the main communication pattern in IS and BT benchmarks is an all to all data exchange. Hence, the analysis of network status graph is identical for pattern based and all-all procedures and the difference is only due to the use of communication skeletons in the pattern based framework.

The broad conclusion from these experiments is that the pattern based approach to node selection offers considerable

improvement in expected performance over random node selection and all-all node selection procedures, but the extent of improvement is strongly dependent on the application characteristics. We should also caution that these are limited experiments and the results will also strongly depend on the network and system characteristics.

4 Discussion

This research employs application characteristics to drive the process of automatically selecting network nodes to execute an application. Specifically, we suggest a two step process for node selection. In the first step, a network status map is matched to the application communication structure to obtain a set of potential node groups for execution. Clearly, better node selection decisions can be made if the procedure is sensitive to application characteristics. In the second step, an application communication skeleton is executed on every candidate group of nodes to decide which group offers the best potential performance. This step is intended to eliminate the various inaccuracies in estimating the communication performance an application can expect on a group of network nodes.

The focus of this paper has been entirely on communication characteristics. We only consider variations in network availability and base node selection on communication capacity. In practice, computation and synchronization considerations are equally important. Computation nodes may have competing loads, and impact of a slowdown in one node in the system may get magnified because of data and control dependencies. In related work, we have addressed the problem of performance estimation with shared nodes and links [17]. However, effective integration and validation of these techniques in a node selection system remains a challenge.

We have only employed communication skeletons, that are a special case of performance skeletons. Construction of complete performance skeletons also includes computation and synchronization considerations. Perhaps the most critical limitation of our system is that the communication skeletons have to be constructed manually. It is not difficult to automatically construct a program that concisely reproduces the measured communication pattern of an application. However, our real goal is automatic construction of general performance skeletons. A performance skeleton should mirror the application it represents in all respects. For example, the computation to communication ratio, synchronization patterns, memory access patterns, fraction of different types of instructions, message exchange patterns, should all be closely correlated between an application and its performance skeleton. The goal is that the relative behavior of the application and performance skeleton should be similar under any computation environment and under

all network conditions. And yet the performance skeleton is expected to execute for a very short time. Note that a performance skeleton cannot be just the beginning part of the application itself since application behavior changes over time and the performance skeleton is expected to capture the cumulative application activities over the full duration of execution. Clearly, automatically constructing performance skeletons is a major challenge, and it is also a key long-term goal of this research. This paper focuses only on demonstrating the value of performance skeletons for application scheduling.

The prototype implementation and results described in this work are essentially a "proof of concept". We point out the most significant limitations of our implementation and experiments. The prototype node selection tool automatically determines the best nodes for execution and schedules the application on those nodes. However, some of the steps in the preprocessing of the applications, to enable them for automatic node selection, are manual. We have conducted experiments on a small compute cluster with the bandwidth controlled to simulate network sharing. More experimentation on larger clusters and grid environments is necessary to evaluate this approach rigorously. Our system currently works only for MPI message passing applications but is not fundamentally limited to any programming model. The NAS benchmark programs used in this research represent a variety of applications in parallel computing, but each benchmark focuses on a single core scientific algorithm. Full applications, in contrast, often employ multiple different types of computations in different phases. This certainly adds additional complexity to node selection that is not evaluated in this work. Overall, we believe that our results are relevant and meaningful, even though there is significant room for more experimentation and better evaluation and validation.

5 Conclusions

This paper makes a case for employing application knowledge to node selection in shared cluster and grid environments. We demonstrate how the communication pattern of an application is exploited to discover good compute nodes and network paths for execution. One of the major problems in automatic node selection for network environments is the cost and accuracy of network usage information. We propose application communication skeletons as our solution approach. With the use of this method, approximate network information is used to get good candidate node groups for execution, and actual execution of skeletons is used to make the final choice. This largely eliminates the potential for poor choices due to inaccurate network information since a small slice of actual execution is performed before assignment of nodes to an application.

We have developed a prototype node selection framework and present results from a network testbed that simulates varying bandwidth availability between pairs of nodes available for execution. The results clearly demonstrate that node selection based on this framework is a large improvement over random node selection, and also a clear improvement over state of the art methods that do not employ application knowledge. While our prototype implementation and experiments are limited in scope, they clearly demonstrate the potential of automated node selection and scheduling based on an application's communication pattern. This paper is a significant step towards general resource scheduling that employs broader application knowledge including computation and synchronization information and general performance skeletons.

6 Acknowledgments

This research was supported in part by the Los Alamos Computer Science Institute (LACSI) through Los Alamos National Laboratory (LANL) contract number 03891-99-23 as part of the prime contract (W-7405-ENG-36) between the DOE and the Regents of the University of California. Support was also provided by the National Science Foundation under award number NSF ACI-0234328 and the University of Houston's Texas Learning and Computation Center.

We wish to thank other current and former members of our research group, in particular, Mala Ghanesh, Amitoj Singh, Sukhdeep Sodhi and Shreenivasa Venkataramaiah, for their contributions to this research.

References

- [1] W. Almesberger. Linux network traffic control — implementation overview. White Paper, April 1999. Available at <ftp://lrcftp.epfl.ch/pub/people/almesber/pub/icio-current.ps>.
- [2] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report 95-020, NASA Ames Research Center, December 1995.
- [3] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing '96*, Pittsburgh, PA, November 1996.
- [4] P. Bhatt, V. Prasanna, and C. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. In *Seventh IEEE Symposium on High-Performance Distributed Computing*, Chicago, IL, July 1998.
- [5] J. Bolliger and T. Gross. A framework-based approach to the development of network-aware applications. *IEEE Trans. Softw. Eng.*, 24(5):376–390, May 1998.
- [6] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-level middleware for the grid. In *Supercomputing 2000*, pages 75–76, 2000.
- [7] I. Foster and K. Kesselman. Globus: A metacomputing infrastructure toolkit. *Journal of Supercomputer Applications*, 11(2):115–128, 1997.
- [8] A. Grimshaw and W. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), January 1997.
- [9] M. Litzkow, M. Livny, and M. Mutka. Condor — A hunter of idle workstations. In *Proceedings of the Eighth Conference on Distributed Computing Systems*, San Jose, California, June 1988.
- [10] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. In *Seventh IEEE Symposium on High-Performance Distributed Computing*, Chicago, IL, July 1998.
- [11] B. Lowekamp, D. O'Hallaron, and T. Gross. Direct queries for discovering network resource properties in a distributed environment. *Cluster Computing*, 3(4):281–291, 2000.
- [12] G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *9th Heterogeneous Computing Workshop*, pages 3–16, 2000.
- [13] A. Singh and J. Subhlok. Reconstruction of application layer message sequences by network monitoring. In *IASTED International Conference on Communications and Computer Networks*, November 2002.
- [14] J. Subhlok, P. Lieu, and B. Lowekamp. Automatic node selection for high performance applications on networks. In *Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 163–172, Atlanta, GA, May 1999.
- [15] J. Subhlok, S. Venkataramaiah, and A. Singh. Characterizing NAS benchmark performance on shared heterogeneous networks. In *11th International Heterogeneous Computing Workshop*, April 2002.
- [16] T. Tabe and Q. Stout. The use of the MPI communication library in the NAS Parallel Benchmark. Technical Report CSE-TR-386-99, Department of Computer Science, University of Michigan, Nov 1999.
- [17] S. Venkataramaiah and J. Subhlok. Performance prediction for simple CPU and network sharing. In *LACSI Symposium 2002*, October 2002.
- [18] J. Weismann. Metascheduling: A scheduling model for metacomputing systems. In *Seventh IEEE Symposium on High-Performance Distributed Computing*, Chicago, IL, July 1998.
- [19] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The Network Weather Service. In *Proceedings of Supercomputing '97*, San Jose, CA, Nov 1997.
- [20] S. Zhou. LSF: load sharing in large-scale heterogeneous distributed systems. In *Proceedings of the Workshop on Cluster Computing*, Orlando, FL, April 1992.

A Performance Study of Job Management Systems*

Tarek El-Ghazawi¹, Kris Gaj², Nikitas Alexandridis¹, Frederic Vroman¹, Nguyen Nguyen²,
Jacek R. Radzikowski², Preeyapong Samipagdi¹, and Suboh A. Suboh¹

¹ The George Washington University, ² George Mason University,
tarek@seas.gwu.edu

Abstract

Job Management Systems (JMSs) efficiently schedule and monitor jobs in parallel and distributed computing environments. Therefore, they are critical for improving the utilization of expensive resources in high-performance computing systems and centers, and an important component of grid software infrastructure. With many JMSs available commercially and in the public domain, it is difficult to choose an optimum JMS for a given computing environment. In this paper, we present the results of the first empirical study of JMSs reported in the literature. Four commonly used systems, LSF, PBS Pro, Sun Grid Engine / CODINE, and Condor were considered. The study has revealed important strengths and weaknesses of these JMSs under different operational conditions. For example, LSF was shown to exhibit excellent throughput for a wide range of job types and submission rates. On the other hand, CODINE appeared to outperform other systems in terms of the average turn-around time for small jobs, and PBS appeared to excel in terms of turn-around time for relatively larger jobs.

1. Introduction

A lot of work has been done in grid software infrastructure. One of the major tasks of this infrastructure is *job management*, also known as workload management, load sharing, or load management. Software systems capable of performing this task are referred to as *Job Management Systems (JMSs)*.

Job Management Systems can leverage under-utilized computing resources in a grid computing like style. Most JMSs can operate in multiple environments, including heterogeneous clusters of workstations, supercomputers, and massively parallel systems. The focus of our study is performance of JMSs in a loosely coupled cluster of heterogeneous workstations.

Taking into account the large number of JMSs available commercially and in public domain, choosing the best JMS for particular type of distributed computing environment is not an easy task. All previous comparisons of JMSs reported in literature had only a conceptual character. In [1], selected JMSs were compared and contrasted according to a set of well defined criteria.

In [2, 3, 4], the job management requirements for the Numerical Aerodynamic Simulation (NAS) parallel systems and clusters at NASA Ames Research Center were analyzed and several commonly used JMSs evaluated according to these criteria. In [5, 6, 7], three widely used JMSs were analyzed from the point of view of their use with Sun HPC Cluster Tools. Finally, our earlier conceptual study, reported in [7, 8, 9], gave a comparative overview and ranking of twelve popular systems for distributed computing, including several JMSs.

In this paper, we extend the conceptual comparison with the empirical study based on a set of well defined experiments performed in a uniform fashion in a controlled computing environment. To our best knowledge, this is a first reported experimental study quantifying the relative performance of several Job Management Systems.

Our paper is organized as follows. In Section 2, we give an introduction to Job Management Systems, and summarize conceptual functional differences among them. In Section 3, we define metrics used for comparison, present our experimental setup, and discuss parameters and role of all experiments. In Section 4, we describe our methodology and tools used for the measurement collection. Finally, in Sections 5 and 6, we present experimental results, their analysis, and we draw conclusions regarding the relative strengths and weaknesses of investigated JMSs.

2. Job Management Systems

2.1. General architecture of a JMS

The objective of a JMS, for an environment investigated in this paper, is to let users execute jobs on a non-dedicated cluster of workstations with a minimum impact on owners of these workstations by using computational resources that can be spared by the owners. The system should be able to perform at least the following tasks:

- a. monitor all available resources,
- b. accept jobs submitted by users together with resource requirements for each job,

*This work has been funded by the DoD under the LUCITE contract #MDA904-98-C-A081 and by the NSF INT-0109038

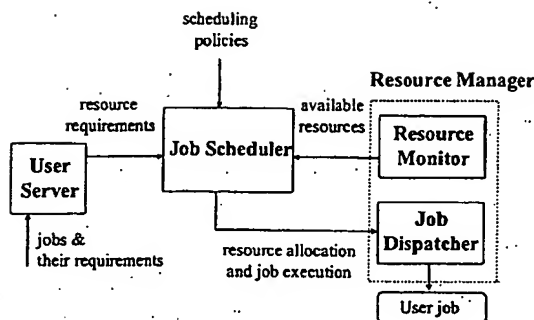


Figure 1. Major functional blocks of a Job Management System

- c. perform centralized job scheduling that matches all available resources with all submitted jobs according to the predefined policies [10],
- d. allocate resources and initiate job execution,
- e. monitor all jobs and collect accounting information.

To perform these basic tasks, a JMS must include at least the following major functional units shown in Fig. 1:

1. *User server* – which lets user submit jobs and their requirements to a JMS (task b), and additionally may allow the user to inquire about the status and change the status of a job (e.g., to suspend or terminate it).
2. *Job scheduler* – which performs job scheduling and queuing based on the resource requirements, resource availability, and scheduling policies (task c).
3. *Resource manager* – used to monitor resources and dispatch jobs on a given execution host (tasks a, d, e).

2.2. Choice of Job Management Systems

More than twenty JMS packages, both commercial and public domain, are currently in use [1, 7]. For interest of time we selected four representative and commonly used JMSs

- LSF – Load Sharing Facility
- PBS – Portable Batch System
- Sun Grid Engine / CODINE, and
- Condor

The common feature of these JMSs is that all of them are based on a central Job Scheduler running on a single node.

LSF (Load Sharing Facility) is a commercial JMS from Platform Computing Corp. [11,12]. It evolved from Utopia system developed at the University of Toronto [13], and is currently probably the most widely used JMS.

PBS (Portable Batch System) has both a public domain and a commercial version [14]. The commercial version called PBS Pro is supported by Veridian Systems. This

version was used in our experiments. PBS was originally developed to manage aerospace computing resources at NASA Ames Research Center.

Sun Grid Engine/CODINE is an open source package supported by Sun Inc. It evolved from DQS (Distributed Queuing System) developed by Florida State University. Its commercial version called CODINE was offered by GENIAS GmbH in Germany and became widely deployed in Europe.

Condor is a public domain software package that was started at University of Wisconsin. It was one of the first systems that utilized idle workstation cycles and supported checkpointing and process migration.

2.3. Functional similarities and differences among selected Job Management Systems

The most important functional characteristics of selected four JMSs are presented and contrasted in Table 1. From this table, it can be seen that LSF supports all operating systems, job types, and features included in the table. CODINE lacks support for Windows NT, stage-in and stage-out, and checkpointing. PBS and Condor trail LSF and CODINE in terms of support for parallel jobs, dynamic load balancing and master daemon fault recovery. They also support a smaller number of operating systems compared to LSF.

3. Experimental Setup

3.1. Metrics

The following performance measures were investigated in our study:

1. **Throughput** is defined in general as a number of jobs completed in a unit of time. Since this number depends strongly on how many jobs are taken into account, we consider *throughput* to be a function of the number of jobs, k , and define it as k divided by the amount of time necessary to complete k JMS jobs (see Fig. 2a). We also define *total throughput* as a special case of throughput for parameter k equal to the total number of jobs submitted to a JMS during the experiment, N (see Fig. 2b).

In Fig. 3, we show the typical dependence of throughput on the number of jobs taken into account, k . It can be seen that throughput increases sharply as a function of k until the moment when either all system CPUs become busy, or the number of jobs submitted and completed in a unit of time become equal. When the

Table 1. Conceptual functional comparison of selected Job Management Systems

	LSF	CODINE	PBS	Condor
Distribution	commercial	public domain	commercial and public domain	public domain
Operating System Support				
Linux, Solaris	yes	yes	yes	yes
Tru64	yes	yes	yes	no
Windows NT	yes	no	no	partial
Types of Jobs				
Interactive jobs	yes	yes	yes	no
Parallel jobs	yes	yes	partial	limited to PVM
Features Supporting Efficiency, Utilization, and Fault Tolerance				
Stage-in and stage-out	yes	no	yes	yes
Process migration	yes	yes	no	yes
Dynamic load balancing	yes	yes	no	no
Checkpointing	yes	using external libraries	only kernel-level	yes
Daemon fault recovery	master and execution hosts	master and execution hosts	only for execution hosts	only for execution hosts

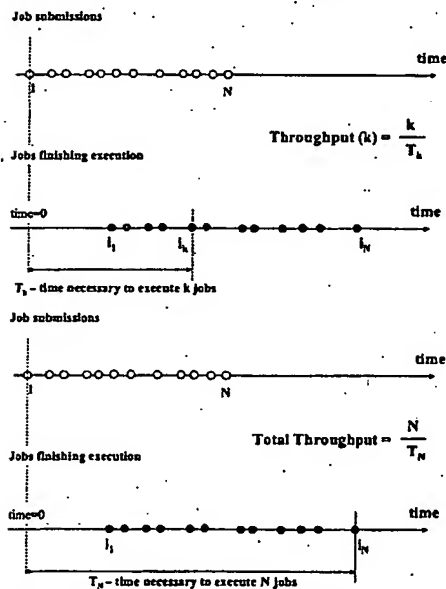


Figure 2. Definition of (a) throughput and (b) total throughput

number of jobs taken into account, k , gets close to the total number of jobs submitted during the experiment, the throughput drops sharply and unpredictably. This drop is the result of a boundary effect and is not likely to appear during the regular operation of a JMS, when the flow of jobs submitted to the JMS continues uninterrupted for a long period of time. Therefore, we decided to use for comparison

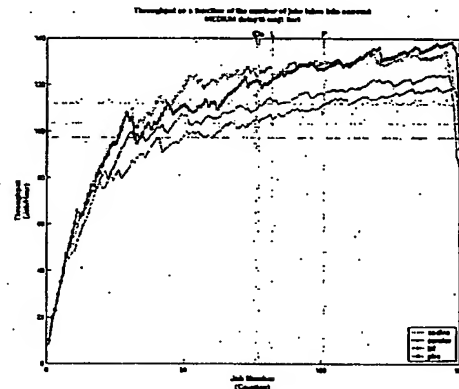


Figure 3. Throughput as a function of number of jobs taken into account

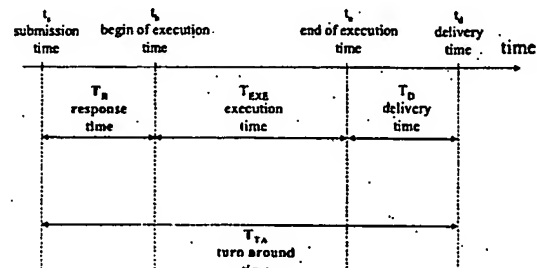


Figure 4. Definition of timing parameters

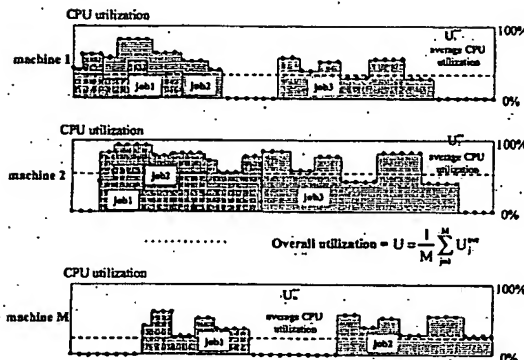


Figure 5. Definition of the system utilization and its measurement using top

average throughput, defined as throughput averaged over all possible values of the job number, k .

2. Average turn-around time is the time from submitting a job till completing it, averaged over all jobs submitted to a JMS (see Fig. 4).

3. Average response time is the average amount of time between submitting a job to a JMS and starting the job on one of the execution hosts (see Fig. 4).

4. Utilization is the ratio of a busy-time span to the available time span. In our experiments, we measured the utilization by measuring the average percentage of the CPU time used by all JMS jobs on each execution host. These average machine utilizations were then averaged over all execution hosts (see Fig. 5).

3.2. Our Micro-grid testbed

A Micro-grid testbed used in our experiments is shown in Fig. 6. The testbed consists of 9 PCs running Linux OS, and 4 workstations Ultra 5, running Solaris 8.

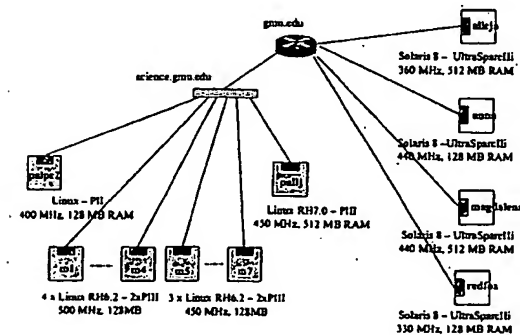


Figure 6. A Micro-grid testbed used in the experimental study

The total number of CPUs available in the testbed is 20. The network structure of the testbed is flat, so that every machine can serve as both an execution host and a submission host. In all our experiments, pallj was used as a master host for all Job Management Systems. All 13 hosts, including the master host, were configured as execution hosts. In all our experiments, pallj was also employed as a submission host.

3.3. Application benchmarks

A set of 36 benchmarks has been compiled and installed on all machines of our testbed. These programs belong to the following four classes of benchmarks: NSA HPC Benchmarks, NAS Parallel Benchmarks, UPC Benchmarks, and Cryptographic Benchmarks. Each benchmark has been characterized in terms of the CPU time, wall time, and memory requirements using one of the Linux machines.

All benchmarks have been divided into the following three sets of benchmarks:

1. Set 1 - Short job list - 16 benchmarks with an execution time between 1 second and 2 minutes, and an average execution time equal to 22 seconds.
2. Set 2 - Medium job list - 8 benchmarks with an execution time between 2 minutes and 10 minutes, and an average execution time equal to 7 minutes 22 seconds.
3. Set 3 - Long job list - 6 benchmarks with an execution time between 10 minutes and 31 minutes, and an average execution time equal to 16 minutes 51 seconds.

3.4. Experiments

Each experiment consists of running N jobs chosen pseudo-randomly from the given set of benchmarks, and submitted one at a time to a given JMS in the pseudo-random time intervals. All jobs were submitted from the same machine, pallj, and belonged to a single user of the system. The rate of the job submissions was chosen to have a Poisson distribution.

The only job requirement specified during the job submission was the amount of available memory. No information about the expected execution time, or limits on the wall or CPU time were specified.

The total number of jobs submitted to a system, N , was chosen based on the expected total time of each experiment, the average execution time of jobs from the given list, and the number of machines in our testbed. In Experiments 1, 3, 4, and 5, regarding short and medium job lists, the total number of jobs was set to 150, which led to a total experiment time of about two hours. In

Table 2. Characteristic features of experiments performed during our study

Experiment Number	Benchmark Set	Average CPU time / Job	Average Time Intervals Between Job Submissions	Total Number of Jobs	Special Assumptions
1	Set 2, Medium job list	7 min 22 s	30 s, 15 s, 5 s	150	one job / CPU
2	Set 3, Long job list	16 min 51 s	2 min, 30 s	75	one job / CPU
3	Set 1, Short job list	22 s	15 s, 10 s, 5 s, 2 s, 1 s	150	one job / CPU
4	Set 2, Medium job list	7 min 22 s	15 s	150	two jobs / CPU
5	Set 1, Short job list	22 s	5 s	150	one job / CPU; emulation of daemon faults

Experiment 2, regarding the long job list, the total number of jobs was reduced to 75 to keep the time of each experiment within the range of 2 hours.

Each experiment was repeated for four JMSes, under exactly the same initial conditions, including the same initial seeds of the pseudo-random generators. Additionally, all experiments were repeated 3 times for the same JMS to minimize the effects of random events in all machines participating in the experiment.

Additionally, each experiment was repeated for several different average job submission rates. These rates have been chosen experimentally in such a way that they correspond to qualitatively different JMS loads. For the smallest submission rate, each system is very lightly loaded. Only a subset of all available CPUs is utilized at any point in time. Any new job submitted to the system can be immediately dispatched to one of the execution hosts. For the highest submission rate, a majority of CPUs are busy all the time, and almost any new job submitted to a JMS must spend some time in a queue before being dispatched to one of the execution hosts.

The characteristic features of five experiments performed during our experimental study are summarized in Table 2. Experiments 1-4 were designed to measure the performance of each JMS for different job submission rates.

Experiment 5 was aimed at quantifying fault tolerance of each JMS by determining its resistance against the master and execution daemon failures. Five minutes after the beginning of this experiment, master daemons on a master host or execution host daemons on a single execution host were killed. In one version of the experiment, the killed daemons were restarted one minute later, in the other version, no further action was taken. In all cases, the total number of jobs that completed execution was recorded and compared with the total number of jobs submitted to the system for execution.

3.5 Common settings of Job Management Systems

An attempt was made to set all JMSes to an equivalent configuration, using the following major configuration settings:

A. Maximum Number of Jobs per CPU

In all experiments, except Experiment 2, a maximum number of jobs assigned simultaneously to each CPU was set to one. In other words, no timesharing of CPUs was allowed. This setting was chosen as an optimum because of the numerical character of benchmarks used in our study. All benchmarks from the short, medium, and long job lists have no input or output. For this kind of benchmarks, timesharing can improve only the response time, but has a negative effect on two most important performance parameters: turn-around time and throughput. This deteriorating effect of timesharing was clearly demonstrated in our Experiment 4, where two jobs were allowed to share the same CPU.

B. CPU factor of execution hosts

The CPU factors determine the relative performance of execution hosts for a given type of load. Based on the recommendations given in the JMS manuals, CPU factors for LSF and CODINE were set based on the relative performance of benchmarks representing a typical load. For each list of benchmarks, two representative benchmarks were selected, and run on all machines of distinctly different types. The CPU factors were set based on an average ratio of the execution time on the slowest machine to the execution time on the machine for which the CPU factor was determined. Based on this procedure, the slowest machine had always a CPU factor equal to 1.0. The CPU factors of remaining machines varied in the range from 1.2 to 1.7 for a small job list, and from 1.4 to 1.95 for the medium and long job lists. The CPU factors of Condor were computed automatically by this JMS based on the Condor-specific benchmarks running on the execution hosts in the spare time. The CPU factors in LSF,

CODINE, and Condor affect the operation of the scheduler. In PBS, the equivalent parameter has no effect on scheduling, and affects only accounting and time limit enforcement.

C. Dispatching interval

The dispatching interval determines how often the JMS scheduler attempts to dispatch pending jobs. This parameter clearly affects an average response time, as well as scheduler overhead. It may also influence the remaining performance parameters.

LSF on one side, and PBS, CODINE, and Condor on the other side use a different definition of this parameter. In all systems, this parameter describes the maximum time in seconds between subsequent attempts to schedule jobs. However, in PBS, CODINE, and Condor the attempts to schedule a job also occur whenever a new job is submitted, and whenever a running batch job terminates. The same is not the case for LSF. On the other hand, LSF has two additional parameters that can be used to limit the time spent by the job in the queue, and thus reduce the response time.

F. Scheduling policies

No changes to the parameters describing scheduling policies were made, which means that the default First Come First Serve (FCFS) scheduling policy was used for all systems. One should be however aware that within this policy, a different ranking of hosts fulfilling the job requirements might be used by different JMSs.

4. Methodology and measurement collection

Each experiment was aimed at determining values of all performance measures defined in Section 3.1. All parameters were measured in the same way for all JMSs, using utilities and mechanisms of the operating systems only.

In particular, timestamps generated using the C function `gettimeofday()`, were used to determine the exact time of a job submission, as well as the begin and end of the execution time. The function `gettimeofday()` gets the current time from the operating system. The time is expressed in seconds and microseconds elapsed since Jan 1, 1970 00:00 GMT. The actual resolution of the returned time depends on the accuracy of the system clock, which is hardware dependent.

The Unix Network Time Protocol (NTP) was used to synchronize clocks of all machines of our Micro-grid. The protocol provides accuracy ranging from milliseconds (on LANs) to tenths of milliseconds (on WANs).

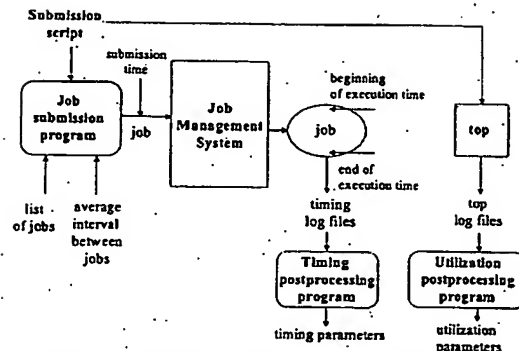


Figure 7. Software used to collect performance measures

In order to determine the JMS utilization, the Unix `top` utility was used. This utility records an approximate percentage of the CPU time used by each process running on a single machine averaged over a short period of time, e.g., 15 seconds (see Fig. 5). For each point in time, the sum of percentages corresponding to all JMS jobs is computed. These sums are then averaged over the entire duration of an experiment, to determine an average utilization of each machine by all JMS jobs. The execution host utilizations averaged over all execution hosts determine the overall utilization of a JMS.

Three programs were developed to support the experiments and were used in a way shown in Fig. 7. A C++ Job Submission program has been written to emulate a random submission of jobs from a given host. This program takes as an input a list of jobs, a total number of submissions, an average interval between two consecutive submissions, and the name of a JMS used in a given experiment. Two post-processing Perl scripts, Timing and Utilization post-processing utilities, have been developed to process log files generated by benchmarks and the `top` utility. These scripts generate exhaustive reports including values of all performance measures separately for every execution host, and jointly for the entire Micro-Grid testbed.

5. Experimental Results

Two most important parameters determining the performance of a Job Management System are turn-around time and throughput. *Throughput* is particularly important when a user submits a large batch of jobs to a JMS and does not do any further processing till all jobs complete execution. *Turn-around time* is particularly important when a user

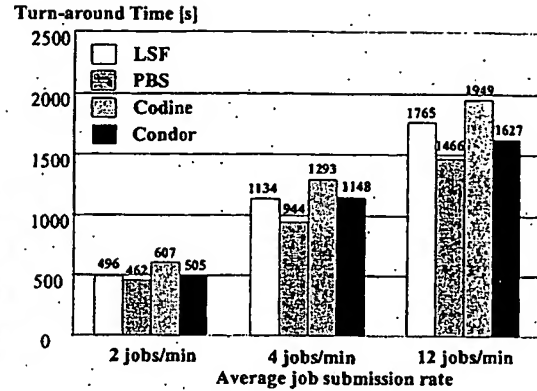
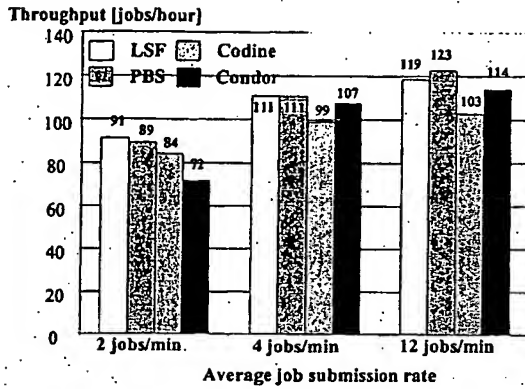


Figure 8. Average throughput and average turn-around-time for the medium job list

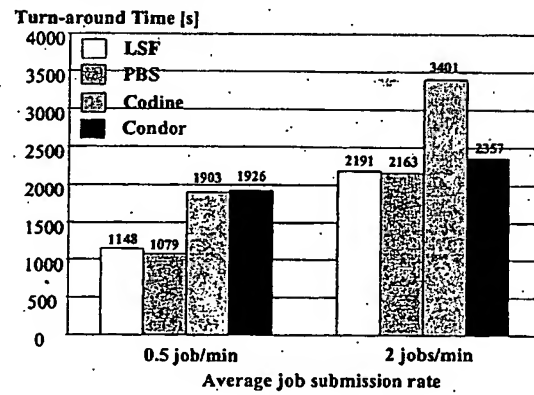
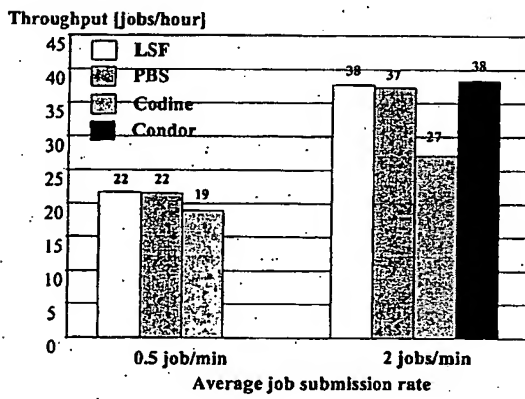


Figure 9. Average throughput and average turn-around-time for the long job list

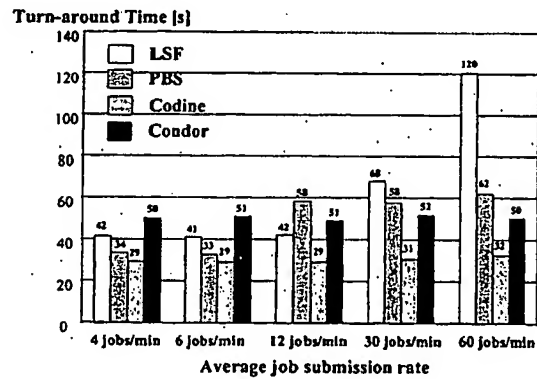
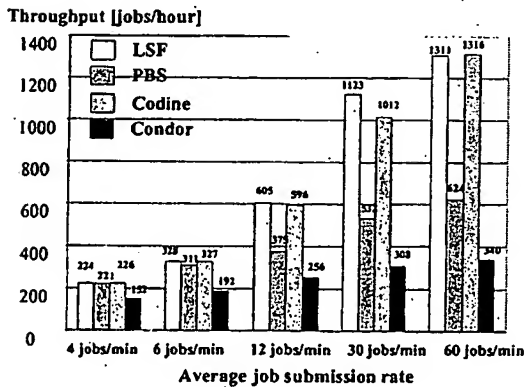


Figure 10. Average throughput and average turn-around-time for the short job list

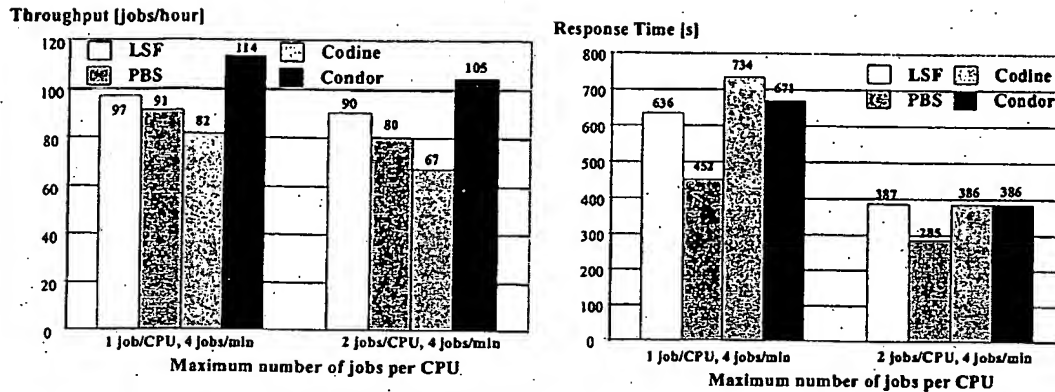


Figure 11. Average partial throughput and response time for medium jobs with two jobs allowed to share a single CPU

Table 3. Results of Experiment 5 regarding daemon fault recovery

JMS	Daemons killed 5 minutes after the beginning of the experiment			
	Master daemons		Execution host daemons	
	Not restarted	Restarted 1 min later	Not restarted	Restarted 1 min later
LSF	Master daemons automatically restarted, no jobs lost	Master daemons automatically restarted before 1 min, no jobs lost.	No jobs lost	No jobs lost
PBS	Daemons did not restart, all later jobs lost	12 jobs submitted during the daemon down time lost	No jobs lost.	No jobs lost
CODINE	Master daemons automatically restarted, no jobs lost	Master daemons automatically restarted before 1 min, no jobs lost.	No jobs lost.	No jobs lost
Condor	Daemons did not restart, all later jobs lost	153 out of 150 jobs finished execution	No jobs lost	No jobs lost

tends to work in a pseudo-interactive mode and awaits results of each subsequent experiment. Additionally, an average response time might be important in case of interactive jobs that require user input and thus a constant presence of the user.

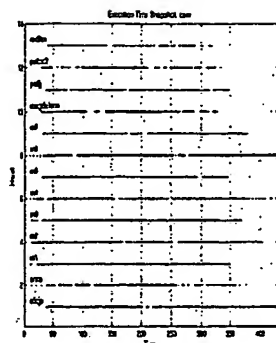
In Experiment 1, with the medium job list, LSF and PBS were consistently the best in terms of the average throughput, while PBS was the best in terms of the average turn-around time. The overall differences among all four systems were smaller than 21% for average throughput, and 33% for average turn-around time.

In Experiment 2, with the long job list, the throughputs of LSF, Condor, and PBS were almost identical, while CODINE was trailing by 29%. The turn-around time was the best for PBS and LSF for both investigated submission rates. For the higher submission rate, the

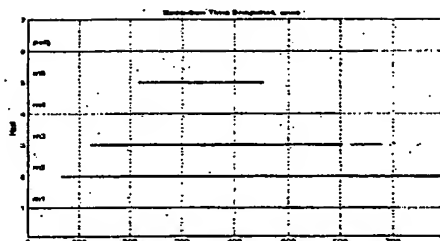
performance of Condor, was approximately the same as performance of two best systems.

In Experiment 3, for short job list, the throughputs of LSF and Codine were the highest of all investigated systems. At the same time, the turn-around time was consistently the best for CODINE.

The analysis of the system utilization and job distribution revealed the following reasons for the different relative performance of each system in terms of throughput and turn-around time. LSF tends to dispatch jobs to all execution hosts, independently of their relative speed (see Fig. 12a). It also uses a complex algorithm for scheduling, which guarantees that jobs are executed tightly one after the other. Both factors contribute to high throughput. At the same time, distributing jobs to all machines, including slow ones, increases average



(a)



(b)

Figure 12. Utilization of machines by a) LSF and b) PBS.

execution time, and complex scheduling affects average response time. Both factors contribute to the increase in the average turn-around time. On the other hand, PBS distributes jobs only to a limited number of the fastest execution hosts (see Fig. 12b). As a result, the average execution time is smaller compared to LSF, which contributes to a better average turn-around time. At the same time, the limited utilization of the execution hosts contributes to only average throughput. Finally, Condor has a scheduling algorithm that seems to be more suitable for longer jobs. Although all execution hosts are utilized, nevertheless, there are significant gaps between the times when one job finishes execution and another job is dispatched to the same execution host.

In Experiment 4, the configuration parameters of each system were changed in order to allow two JMS jobs to execute on each execution host at the same time. Since all jobs used in our study are numerical, and have limited input/output, timesharing of jobs could not improve either average throughput or average turn-around time. In fact, these parameters deteriorated by a factor of 7 to 18%, depending on the JMS, because of the redundancy associated with context swapping. The only parameter that improved was the average response time that decreased by a factor ranging from 1.6 to 1.9 times.

In Experiment 5 (see Table 3), LSF was shown to be resistant against the master daemon failure. When the master daemons were killed, they were automatically restarted shortly after. No jobs were lost. PBS and Condor, do not have the capability to restart killed master daemons but they resume normal operation after these daemons are restarted manually. The interruption has a limited effect on the number of jobs that complete execution. All JMSs appeared to be resistant against the failure of execution host daemons. These daemons were not automatically restarted, but as a result of their failure, affected jobs were redirected to other execution hosts.

6. Summary and Conclusions

The summary of the performance of all investigated Job Management Systems as a function of the job size and the job submission rate is given in Tables 4 and 5.

Table 4. JMS ranking in terms of the average throughput - summary (B - relatively best performance, W - relatively worst performance)

Job size	Submission rate		
	Low	Medium	High
Large	B: LSF, PBS W: CODINE (-14%)	B: Condor, LSF, PBS W: CODINE (-29%)	
Medium	B: LSF, PBS W: CODINE (-21%)	B: LSF, PBS W: CODINE (-11%)	B: PBS, LSF W: CODINE (-17%)
Small	B: LSF, CODINE W: Condor (-57%)	B: LSF, CODINE W: Condor (-69%)	B: LSF, CODINE W: Condor (-71%)

Table 5. JMS ranking in terms of the average turn-around time - summary (B - relatively best performance, W - relatively worst performance)

Job size	Submission rate		
	Low	Medium	High
Large	B: PBS, LSF W: Condor, CODINE (+78%)	B: PBS, LSF W: CODINE (+57%)	
Medium	B: PBS W: CODINE (+31%)	B: PBS W: CODINE (+37%)	B: PBS W: CODINE (+33%)
Small	B: CODINE W: Condor (+72%)	B: CODINE W: PBS (+119%)	B: CODINE W: LSF (+275%)

Based on Tables 4 and 5, and Figures 8-11, we can draw the following conclusions. For large jobs with medium submission rate, Condor has compared favorably with the rest of the systems. In terms of the average system throughput, LSF appears to offer the best

performance for all job sizes and submission rates. In terms of the average turn-around time, PBS is the best for large and medium jobs, but CODINE outperforms it for short jobs.

The relative performance of Job Management Systems was similar for medium and large jobs, and changed considerably for short jobs where the job execution times became comparable with the times required for resource monitoring and job scheduling. CODINE appeared to be particularly efficient for small jobs, while the relative performance of PBS and Condor improved with the increase in the job size and the job submission rate. LSF was the only system that performed uniformly well for all job sizes and submission rates with the exception of the turn-around time for small jobs and large submission rates.

Despite the limitations resulting from the relatively small size of our Micro-Grid testbed and a limited set of system settings exercised in our experiments, the practical value of our empirical knowledge comes, among the other, from the following factors:

- Even though our benchmarks and experiment times seem to be relatively short compared to the real-life scenarios, we make up for that by setting the average time between job submissions to the relatively small values. As a result, the systems are fully exercised, and our results are likely to scale for more realistic loads with proportionally longer job execution times and longer times between job submissions.
- Typical users rarely use all capabilities of any complicated system, such as JMS. Instead, majority of Job Management Systems deployed in the field use the default values of majority of configuration parameters.

Additionally, to our best knowledge, our study is the first empirical study of Job Management Systems reported in the literature. Our methodology and tools developed as a result of this project may be used by other groups to extend the understanding of similarities and differences among behavior and performance of existing Job Management Systems.

References

- [1] M. A. Baker, G. C. Fox, and H. W. Yau, "Cluster Computing Review," NPAC Technical Report SCCS-748, Northeast Parallel Architectures Center, Syracuse University, Nov. 1995.
- [2] J. P. Jones, "NAS Requirements Checklist for Job Queuing/Scheduling Software," NAS Technical Report NAS-96-003 April 1996.
- [3] J. P. Jones, "Evaluation of Job Queuing/Scheduling Software: Phase 1 Report," NAS Technical Report, NAS-96-009, September 1996.
- [4] M. Papakhian, "Comparing Job-Management Systems: The User's Perspective," IEEE Computational Science & Engineering, vol.5, no.2, April-June 1998.
- [5] C. Byun, C. Duncan, and S. Burks, "A Comparison of Job Management Systems in Supporting HPC Cluster Tools," Proc. SUPERG, Vancouver, Fall 2000.
- [6] O. Hassaine, "Issues in Selecting a Job Management Systems (JMS)," Proc. SUPERG, Tokyo, April 2001.
- [7] T. El-Ghazawi, Gaj, Alexandridis, Schott, Staicu, Radzikowski, Nguyen, and Suboh, *Conceptual Comparative Study of Job Management Systems*, Technical Report, February 2001.
<http://ece.gmu.edu/lucite/reports.html>
- [8] T. El-Ghazawi, Gaj, Alexandridis, Schott, Staicu, Radzikowski, Nguyen, Vongsaard, Chauvin, Samipagdi, and Suboh, *Experimental Comparative Study of Job Management Systems*, Technical Report, July 2001.
<http://ece.gmu.edu/lucite/reports.html>
- [9] A. V. Staicu, J. R. Radzikowski, K. Gaj, N. Alexandridis, and T. El-Ghazawi, "Effective Use of Networked Reconfigurable Resources," Proc. 2001 MAPLD Int. Conf., Laurel, Maryland, Sep. 2001.
- [10] Rajesh Raman, Miron Livny, and Marvin Solomon, "Resource Management through Multilateral Matchmaking", Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, PA., August 2000.
- [11] I. Lumb, "Wide-area parallel computing: A production-quality solution with LSF," Supercomputing 2000, Dallas, Texas, Nov. 2000.
- [12] LSF, <http://www.platform.com/products/LSF/>
- [13] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: A load sharing facility for large heterogeneous distributed computer systems," *Software Practice and Experience*, vol.23, no.12, Dec. 1993.
- [14] Portal Batch System (PBS), <http://pbspro.com>

How Sun™ Grid Engine, Enterprise Edition 5.3 Works

This document shows how Sun™ Grid Engine, Enterprise Edition (SGEEE) software policies function to create a productive compute intensive environment. It explains how SGEEE software provides solutions for executives and technical staff involved in implementing projects that require the delivery of abundant compute power in an enterprise setting.

How This Document Is Organized

The first section provides an overview of the SGEEE software computing environment. Then it explains the function of policies in the SGEEE software environment. Finally, the document gives examples of how these policies can be applied to channel compute resources efficiently in the enterprise environment.

This document refers to Sun Grid Engine, Enterprise Edition only. Complete product documentation, as well as other product information, including information about Sun Grid Engine (Standard Edition) is available at

<http://www.sun.com/gridware>

What Is Grid Computing?

Conceptually, a Grid is quite simple—it is a collection of computing resources that perform tasks. It appears to users as a large system, providing a single point of access to powerful distributed resources. Users treat the Grid as a single computational resource. Resource management software, such as Sun Grid Engine Enterprise Edition, accepts jobs submitted by users and schedules them for execution on appropriate systems in the Grid based upon resource management policies. Users can literally submit thousands of jobs at a time without being concerned about where they run.

No two grids are alike; one size **does not** fit all situations. There are three key classes of grids, which scale from single systems to supercomputer-class compute farms that utilize thousands of processors:

- *Cluster Grids* are the simplest, consisting of one or more systems working together to provide a single point of access to users in a single project or department.
- *Campus Grids* enable multiple projects or departments within an organization to share computing resources. Organizations can use campus grids to handle a wide variety of tasks, from cyclical business processes to rendering, data mining, and more.
- *Global Grids* are a collection of campus grids that cross organizational boundaries to create very large virtual systems. Users have access to compute power that far exceeds the resources available within their own organization.

Sun Grid Engine Enterprise Edition (SGEEE) v5.3 Beta software, the newest version of Sun's resource management software solution, provides the power and flexibility required for Campus Grids. SGEEE software orchestrates the delivery of computational power based upon enterprise resource policies set by the organization's technical and management staff. SGEEE software uses these policies to examine the available computational resources within the Campus Grid, gathers these resources, and then allocates and delivers them automatically in a way that optimizes usage across the Campus Grid.

To enable cooperation within the Campus Grid, project owners using the Grid need to negotiate policies; have flexibility in the policies for manual overrides for unique project requirements; and have the policies automatically monitored and enforced. For example, SGEEE can allocate compute cycles to a project with an immediate deadline. One automobile manufacturer uses SGEEE for running car crash simulation projects. A government agency runs up to 24,000 simultaneous jobs to produce their monthly budget reports.

Policy Overview

Policies are determined by the particular needs of the organization at any moment; they are implemented by the compute farm system administrator. There are four policy systems in SGEEE software.

- The share based or share tree policy
- The functional policy
- The override policy
- The deadline policy

Share Based Or Share Tree Policy

The share based policy allocates a percentage of compute resources to all defined users of the compute farm: all the users *share* the resources as described in the cases above. In addition, the share policy addition adjusts for past usage of resources. In case a user accumulates more resource usage than due by the user's entitlement defined in the share tree, SGEEE software adjusts for this "over-usage" by lowering the entitlement of that user for a certain period of time until the user's resource usage meets the allocated entitlement. Conversely, the user's entitlement might need to be increased temporarily, to compensate for "under-utilization" of resources in the past.

Override Policy

The override policy is the straightforward assignment of tickets to a user (or a project or other categories) by the system administrator, altering the usage of the compute resources in the direction desired. Unlike deadline policy tickets, which are automatically withdrawn after the user's job executes, override policy tickets stay with the user until withdrawn manually by the system administrator.

Functional Policy

The functional policy is similar to the share tree policy, but it has no penalties or compensation based on past usage. It functions like a particular case of the share tree policy where the half-life is zero and the compensation factor is 1.

For performance reasons, the functional policy is not implemented as a special case of the share based policy. Therefore, it does not present itself as a simplified share tree in the GUI. By activating both the share tree and the functional policies, a user could receive tickets from both policies, making the actual share distribution (not the

entitlements) less easy to track. Because of that, it is wise not to mix both share and functional policies during an initial installation at a new site. This policy can be activated at a later time during fine tuning.

Deadline Policy

The deadline policy allocates a large number of tickets to a user at a specified time when a job needs to execute. The number of tickets increases linearly up to the maximum number of tickets assigned to the user. This policy will take away the extra tickets from the user after the job finishes. As with the functional policy, it is advisable to activate a deadline policy as a separate step after the basic installation is complete. A deadline policy is really a fine-tuning of the system by an experienced system administrator. When assigning tickets with a deadline policy, the system administrator must be experienced enough to know how many deadline tickets need to be allocated so the job starts in time to meet the deadline.

The Share Based Policy: An Example

To set up a share based policy, we have to calculate the percentage of the total enterprise's compute power (the compute farm) that a user or a department or a project is entitled to receive. This calculation is the *share tree*. In the example that follows, we will set up a SGEEE software policy based on a share tree.

(The figures used in these examples are arbitrarily decided - they are used only to show how a share tree policy works.)

We calculated the amount of the share by starting from the root of the share tree shown in Figure 1.

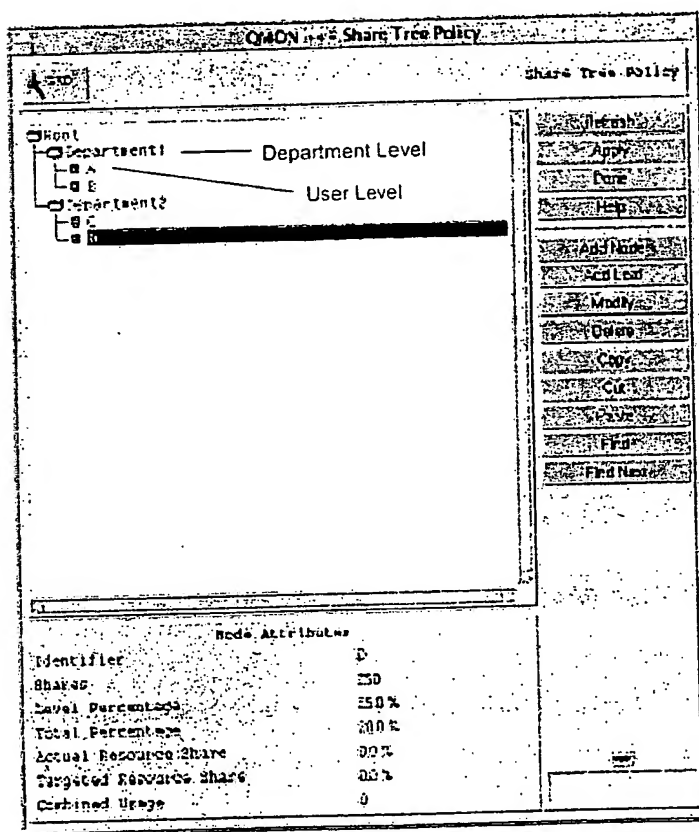


FIGURE 1 Levels at the root of the share tree

To calculate the share based policy, we need to determine the ratio of shares at each "level" of usage. In the example provided, there are two levels, the departmental level and the user level. Then, the number of shares can be assigned by the system administrator.

Department #1: has 200 shares assigned

Department #2: has 800 shares assigned

At the departmental level, there are 1000 shares. Department #1 has a 20% share entitlement; Department #2 has an 80% share entitlement.

At the next level, in this example the user level, user shares are assigned for each user.

Department 1

Department #1 has 2 users (or projects), User A and User B.

User A: has 500 shares assigned

User B: has 500 shares assigned

Each user in Department #1 has a 50% share entitlement of the shares available to the department.

In terms of the total number of shares in the grid cluster, User A has 10% share entitlement of the total resources in the cluster. [20% entitlement of Department #1's shares * 50% entitlement of User A at the departmental level] User B also has 10% share of the total resources available to the cluster.

Department 2

Department #2 also has 2 users, User C and User D.

User C: has 750 shares assigned

User D: has 250 shares assigned

User C has a 75% share entitlement of Department #2. User D has a 25% share entitlement of Department #2.

In terms of the total number of shares in the grid cluster, User C has 60% share entitlement of the total resources in the cluster. [80% entitlement of Department #2's shares * 75% entitlement of User C at the departmental level] User D has a 20% share of the total resources available to the cluster.

This first share distribution used is shown at the top of Figure 2, Entering Tickets.

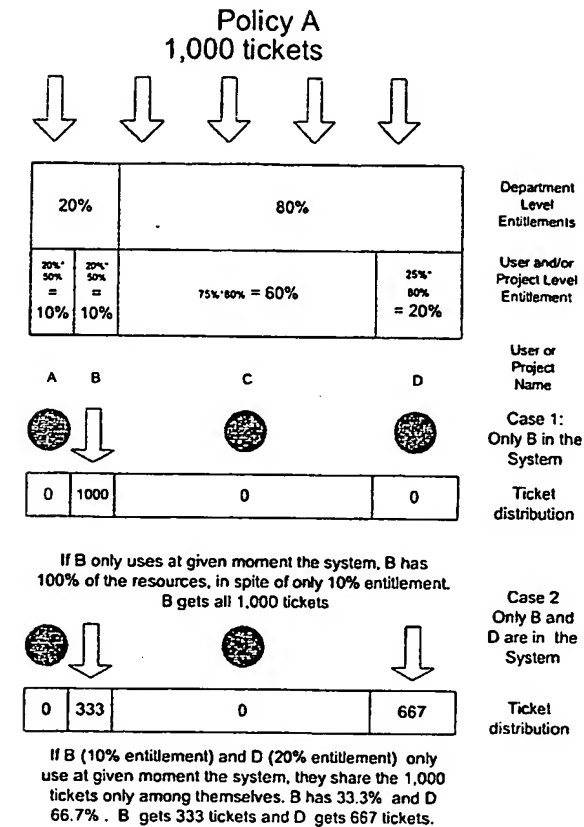


FIGURE 2 Entering Tickets

In the grid cluster, work, in the form of a compute job, is called a ticket. Once work is requested by a user (or users) within the grid cluster, computational resources are assigned according to the share entitlement policy.

If 1000 tickets are assigned to this grid cluster by the system administrator, the tickets will be distributed to the grid cluster in a manner consistent with the entitlements of the share tree, the share policy set for this grid cluster.

Current Active Tickets		Edit Tickets	
Share Tree Tickets	1000	Total Share Tree Tickets	1000
Functional Tickets	0	Total Functional Tickets	0
Deadline Initiation Tickets	0	Maximum Deadline Initiation Tickets	0
Override Tickets	0		

Buttons: Refresh, Apply, Done, Help

Policy: Users Allowed, Share Tree Policy, Functional Policy, Override Policy

Success

FIGURE 3 Assigning tickets to a policy.

The tickets are distributed to all users or projects active at any given time according to the share policy. The following three cases illustrate the allocation of compute resources according to this share tree entitlement.

Case 1: Only User B Is Active.

In case 1, User B is the only active user on the grid cluster. User B has an entitlement of 10%, but because there are no other users submitting tickets, User B has 100% of the total 1000 active tickets. In case 1, User B receives all 1000 tickets for compute resources. See "Entering Tickets" on page 7.

Case 2: Users B And D Are Active.

In case 2, both User B and User D are active. User B has 33.3% actual usage, still much higher than the assigned entitlement of 10%. User D has 66.6% actual usage, much higher than the assigned entitlement of 20%. In this case, the 1,000 tickets are distributed as follows: User B gets 333 tickets and User D gets 667 tickets. Note that Sun Grid Engine, Enterprise Edition does not allow fractional tickets. Again, See "Entering Tickets" on page 7.

Case 3: Adding Override Policy Tickets

For case 3, assume that only Users B and D are active but that User B needs more compute resources for a limited time. In case 3, the system administrator issues a different policy, an Override Policy. The override policy in this case is 700 override tickets for a week.

During that week, SGE software has active 2 policies, 1) the share based policy with 1000 departmental tickets and 2) the override policy with 700 tickets, assigned 100% to User B. In Case 3, there are 1700 tickets in circulation as long as User B is active.

Assuming that User A and User C are inactive for these seven days, User B has 333 share based tickets and additional 700 policy override tickets for a total of 1033 tickets.

During this time, User D has 667 share policy tickets.

As a percentage of the total compute power resources, User B has 60.8% (1033/1700) of the available resources (up from 33.3%) and User D has 39.2% (667/1700) of the available resources (down from 66.7%).

The screenshot shows a window titled "QMON Override Policy". Inside, there is a "GRD" button and a "User" dropdown menu. To the right of the dropdown are buttons for "Refresh", "Apply", "Done", and "Help". Below the dropdown is a table with two columns: "User" and "Tickets". The table contains the following data:

User	Tickets
C	0
A	0
B	1000
D	0

At the bottom right of the window, there is a "Success" message.

FIGURE 4 The Override Policy with 700 tickets assigned to User B.

The Half-life Factor

Half-life is how fast the system forgets about a user's resource consumption. The system administrator can decide whether to or how to penalize a user for high resource consumption, be it six months ago or six days ago. On each node of the share tree, Sun Grid Engine, Enterprise Edition maintains a record of users' resource consumption.

With this record, the system administrator can decide how far to look back to determine a user's under/over-utilization when setting up a share based policy. The resource usage in this context is a mathematical integral (sum) of all the computer resources consumed over a "sliding window of time." The length of this window is determined by a "half-life" factor, which in Sun Grid Engine, Enterprise Edition is an internal decay function. This decay function reduces the impact of accrued resource consumption over time. A short half-life quickly lessens the impact of resource over-consumption; a longer half-life gradually lessens the impact of resource over-consumption.

In SGEEE software this half-life decay function is a specified unit of time. For example, a half-life of 7 days applied to a resource consumption of 1000 units results in the following usage "penalty" adjustment over time.

- . 500 after 7 days
- . 250 after 14 days
- . 125 after 21 days
- . 62.5 after 28 days

The half-life based decay diminishes the impact of a user's resource consumption over time, until the penalization effect is very small and negligible. The exact description of the decay function is shown in the Appendix 1. Note that if a user receives override tickets, these are not subjected to a past usage penalty as they belong to a different policy system. The decay function is a characteristic of the share tree policy only.

The Compensation Factor

In a share based policy system, it is also possible to compensate for a user's inactivity. The administrator has control over this situation via the so called "compensation factor". This factor defines a short-term entitlement adjustment for a user who was previously inactive. This adjustment compensates the user automatically by an amount relative to that user's long term entitlement defined by the share tree policy.

For example, if the entitlement defined by the share tree policy for User B is 10% and the compensation factor assigned to this user is 5, then the system can raise user B's short term entitlement up to 50%.

FIGURE 5 Typical half-life and compensation factor set up

Case 4: Half-life And Compensation Factors

This case is a modification of Case 2. The original share entitlements for Case 2 were as follows:

User A = 10%

User B = 10%

User D = 20%

User C is inactive

The total resource entitlement for these three users A,B and C is 40%. In this case, the 1000 tickets are distributed as follows:

User A = $1,000 \cdot 10\% \cdot (100/40) = 250$ tickets = 25% (current)

User B = $1,000 \cdot 10\% \cdot (100/40) = 250$ tickets = 25% (current)

User D = $1,000 \cdot 20\% \cdot (100/40) = 500$ tickets = 50% (current)

Now assume User A starts using the system after User B and User D already utilized the cluster for some time.

In this case, User B and User D have accumulated more resources than they were entitled to because User C has been inactive for some time. For example, User B might have consumed 33% of the resources and User D 66% of the resources. Meanwhile, user User C consumed less than 1%. Sun Grid Engine, Enterprise Edition could dynamically adjust the entitlements for a short time so that User A gets 70% of the resources, while User B receives 10% and User D receives 20%.

However, if a compensation factor of 5 is assigned to User A, the most resource usage User A could be allocated would be 5 times the 10% share policy entitlement, a maximum of 50% resource allocation.

In this case, User B might get 17% of the available resources and User D would receive 33%.

User A with 50%, up from 25%

User B with 17%, down from 25%

User D with 33%, down from 50%.

Note that this short term entitlement adjustment will change dynamically as User C accumulates resource usage according to the long term entitlement defined by the share tree policy.

Defining Compute Resources In Grid, Enterprise Edition

As part of the share tree policy, Sun Grid Engine, Enterprise Edition software allows the system administrator to assign factors of relative importance among three types of compute resources: CPU cycles, I/O activity, and memory usage. These three resources are defined as follows:

- CPU cycles, as measured by the operating system
- Memory, measured as a mathematical integral over time
(2 GB over 2 days equals 4GB for 1 day)
- I/O as reported by the operating system

For example, if only the CPU cycle is considered as a compute resource, the system administrator will assign a value of 100% to the CPU and zero to the other two. Using three sliders in the GUI, adjustments in resource allocation can be made.

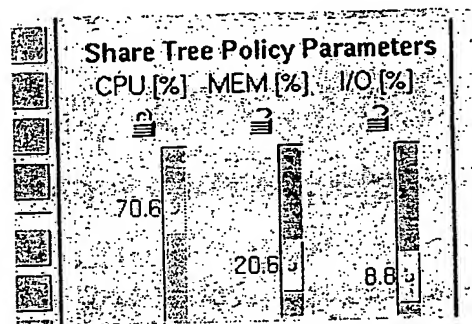


FIGURE 6 Usage definition in the share tree GUI

Sun Grid Engine, Enterprise Edition software can account for accumulated consumption according to these definitions and can compare it with the entitlement of users and projects in the share tree policy to calculate consumption penalties.

Priorities in SGEEE Software

The Sun Grid Engine, Enterprise Edition scheduler dispatches jobs only according to the ratios of the tickets in circulation at a given moment in time. There is no priority setting in Sun Grid Engine, Enterprise Edition software. This can create the following behavior.

Assume Department A has 9,000 tickets and 100 jobs submitted. Each job has 90 tickets.

If Department B has 1,000 tickets, but only 4 jobs, each job will have 250 tickets. The jobs from Department B may then execute before the jobs from Department A which has only 90 tickets, in spite of the fact that Department A has an entitlement to 90% of the resources.

This behavior will correct itself, however, as Department B accumulates resources above its entitlements and therefore will be "punished" with an adjusted lower and lower temporary entitlement. This will lead to less and less tickets per Department B jobs and thus will allow Department A jobs to get scheduled.

Appendix 1: Mathematical Definition Of the Decay Factor

The actual formula used to define SGEEE software's decay factor is derived from nuclear physics

Radioactive decay is described by the following formula:

$$N(t) = N(0) * \exp(-D*t)$$

$N(t)$ is the number of particles at time t . $N(0)$ is the initial number. D is the decay rate, a material constant, and t is time. If you know the half-life $t(H)$, then with this formula you can compute D :

$$D = -\log(1/2)*t(H)$$

With this D , for any fixed time interval you can compute a constant factor (evaluating the e-function using the constant D and t values) reducing a number of particles at the beginning of the interval into the resulting (not decayed) number of particles at the end of the interval. If we call this factor R , we have:

$$N(end) = N(begin) * R \quad (\text{Note: } 0 < R < 1)$$

You can use this iteratively with $N(end)$ equal to $N(begin)$ in the succeeding interval.

This is exactly what is done in SGEEE software. With the half-life time factor specified by the administrator, R is computed for the GRD scheduler interval. Then during every scheduler interval, all the accumulated resource consumption for each user and project is decayed by the simple multiplication above.

Of course, at each scheduler interval, additional "new" usage may be added to the accumulated usage of any user/project. This is not problem, though, as it is "fresh" so to speak and hence not decayed. This fresh usage will be decayed for the first time in the next scheduler interval together with the other accumulated usage by applying the multiplication formula above to the current sum of usage for each user or project. The "older" usage contribution is, (the longer ago it has been added) the more often it will have been multiplied (decayed) and the smaller its impact will be in determining any consumption penalty.

© 2001 Sun Microsystems, Inc. All rights reserved.

TRADEMARKS

Sun, Sun Microsystems, and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.